Application Services Framework Reference

Carbon

2007-10-31

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures and Tables

Introduction

Framework	/System/Library/Frameworks/ApplicationServices
Header file directories	/System/Library/Frameworks/ApplicationServices.framework/Headers
Declared in	AEDataModel.h AEHelpers.h AEInteraction.h AEMach.h AEObjects.h AEPackObject.h AERegistry.h AEUserTermTypes.h ATSFont.h ATSLayoutTypes.h ATSTypes.h ATSUnicodeDirectAccess.h ATSUnicodeDrawing.h ATSUnicodeFlattening.h ATSUnicodeFonts.h ATSUnicodeGlyphs.h ATSUnicodeObjects.h ATSUnicodeTypes.h AXActionConstants.h AXAttributeConstants.h AXNotificationConstants.h AXRoleConstants.h AXValueConstants.h AppleEvents.h CABase.h CGAffineTransform.h CGBitmapContext.h CGColor.h CGColorSpace.h CGContext.h CGDataConsumer.h CGDataProvider.h CGDirectDisplay.h CGDirectPalette.h CGDisplayConfiguration.h CGDisplayFade.h CGError.h CGEvent.h CGEventSource.h CGEventTypes.h CGFont.h CGFunction.h CGGLContext.h

CGGeometry.h
CGGradient.h
CGImage.h
CGImageDestination.h
CGImageProperties.h
CGImageSource.h
CGLayer.h
CGPDFArray.h
CGPDFContentStream.h
CGPDFContext.h
CGPDFDictionary.h
CGPDFDocument.h
CGPDFObject.h
CGPDFOperatorTable.h
CGPDFPage.h
CGPDFScanner.h
CGPDFStream.h
CGPDFString.h
CGPSConverter.h
CGPath.h
CGPattern.h
CGRemoteOperation.h
CGSession.h
CGShading.h
CGWindowLevel.h
CMApplication.h
CMCalibrator.h
CMDeviceIntegration.h
CMICCProfile.h
CMMComponent.h
CMScriptingPlugin.h
CMTypes.h
CarbonEvents.h
Dictionary.h
Displays.h
FindByContent.h
FontSync.h
Fonts.h
HIAccessibility.h
IOMacOSTypes.h
Icons.h
IconsCore.h
ImageCompression.k.h
InternetConfig.h
LanguageAnalysis.h
MacTypes.h
PMCore.h
PMCoreDeprecated.h
PMDefinitions.h
PMDefinitionsDeprecated.h
PMTemplate.h
PMTicket.h
PMTicketDeprecated.h

Palettes.h
Pasteboard.h
PictUtils.h
Processes.h
QDOffscreen.h
QDPictToCGContext.h
QuickTimeComponents.k.h
Quickdraw.h
QuickdrawAPI.h
QuickdrawText.h
QuickdrawTypes.h
SpeechSynthesis.h
X.h

This collection of documents provides the API reference for the Application Services framework, which includes several services—such as Quartz 2D and Mac OS X accessibility features—that are essential to Carbon applications. (The Quartz 2D API is defined in the Core Graphics interfaces, which begin with the letters "CG.") The Application Services framework also includes support for a number of legacy technologies—such as QuickDraw and the Font Manager—that have been superseded with newer technologies like Quartz 2D and ATSUI.

Introduction

Opaque Types

CGBitmapContext Reference

Derived From:	`CGContextRef` (page 137)
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGBitmapContext.h
Companion guide	Quartz 2D Programming Guide

Overview

The CGBitmapContext header file defines functions that create and operate on a Quartz bitmap graphics context. A bitmap graphics context is a type of `CGContextRef` (page 137) that you can use for drawing bits to memory. The functions in this reference operate only on Quartz bitmap graphics contexts created using the functionCGBitmapContextCreate (page 24).

The number of components for each pixel in a bitmap graphics context is specified by a color space (defined by a `CGColorSpaceRef` (page 56), which includes RGB, grayscale, and CMYK, and which also may specify a destination color profile). The bitmap graphics context specifies whether the bitmap should contain an alpha channel, and how the bitmap is generated.

Functions by Task

Creating Bitmap Contexts

`CGBitmapContextCreate` (page 24)
Creates a bitmap graphics context.

`CGBitmapContextCreateImage` (page 25)
Creates and returns a Quartz image from the pixel data in a bitmap graphics context.

Getting Information About Bitmap Contexts

These functions return the values of attributes specified when a bitmap context is created.

`CGBitmapContextGetBitmapInfo` (page 26)
Obtains the bitmap information associated with a bitmap graphics context.

`CGBitmapContextGetAlphaInfo` (page 26)
Returns the alpha information associated with the context, which indicates how a bitmap context handles the alpha component.

`CGBitmapContextGetBitsPerComponent` (page 27)
Returns the bits per component of a bitmap context.

`CGBitmapContextGetBitsPerPixel` (page 27)
Returns the bits per pixel of a bitmap context.

`CGBitmapContextGetBytesPerRow` (page 28)
Returns the bytes per row of a bitmap context.

`CGBitmapContextGetColorSpace` (page 28)
Returns the color space of a bitmap context.

`CGBitmapContextGetData` (page 28)
Returns a pointer to the image data associated with a bitmap context.

`CGBitmapContextGetHeight` (page 29)
Returns the height in pixels of a bitmap context.

`CGBitmapContextGetWidth` (page 29)
Returns the width in pixels of a bitmap context.

Functions

CGBitmapContextCreate

Creates a bitmap graphics context.

```
CGContextRef CGBitmapContextCreate (
   void *data,
   size_t width,
   size_t height,
   size_t bitsPerComponent,
   size_t bytesPerRow,
   CGColorSpaceRef colorspace,
   CGBitmapInfo bitmapInfo
);
```

Parameters

data
A pointer to the destination in memory where the drawing is to be rendered. The size of this memory block should be at least `(bytesPerRow*height)` bytes.

Starting in Mac OS X v10.3, you can pass `NULL` if you don't care where the data is stored. This frees you from managing your own memory, which reduces memory leak issues. Quartz has more flexibility when it manages data storage for you. For example, it's possible for Quartz to use OpenGL for rendering if it takes care of the memory.

width
The width, in pixels, of the required bitmap.

height
The height, in pixels, of the required bitmap.

bitsPerComponent
The number of bits to use for each component of a pixel in memory. For example, for a 32-bit pixel format and an RGB color space, you would specify a value of 8 bits per component. For more information about supported pixel formats, see *Quartz 2D Programming Guide*.

bytesPerRow
The number of bytes of memory to use per row of the bitmap.

colorspace
The color space to use for the bitmap context. Note that indexed color spaces are not supported for bitmap graphics contexts.

bitmapInfo
A `CGBitmapInfo` constant that specifies whether the bitmap should contain an alpha channel and its relative location in a pixel, along with whether the components are floating-point or integer values. (See *CGImage Reference* for a description `CGBitmapInfo` constants.) In *Quartz 2D Programming Guide*, see "Creating a Bitmap Graphics Context" (in the *Graphics Contexts* chapter) for the color space, bits per pixel, bits per pixel component, and bitmap information constant combinations that you can use when creating a bitmap context with `CGBitmapContextCreate`.

Return Value
A new bitmap context, or `NULL` if a context could not be created. You are responsible for releasing this object using `CGContextRelease` (page 102).

Discussion
When you call this function, Quartz creates a bitmap drawing environment—that is, a bitmap context—to your specifications. When you draw into this context, Quartz renders your drawing as bitmapped data in the specified block of memory.

The pixel format for a new bitmap context is determined by three parameters—the number of bits per component, the color space, and an alpha option (expressed as a `CGBitmapInfo` (page 227) constant). The alpha value determines the opacity of a pixel when it is drawn.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGBitmapContext.h`

CGBitmapContextCreateImage

Creates and returns a Quartz image from the pixel data in a bitmap graphics context.

```
CGImageRef CGBitmapContextCreateImage (
   CGContextRef c
);
```

Parameters

c
A bitmap graphics context.

Return Value
A CGImage object that contains a snapshot of the bitmap graphics context or `NULL` if the image is not created.

Discussion
The CGImage object returned by this function is created by a copy operation. Subsequent changes to the bitmap graphics context do not affect the contents of the returned image. In some cases the copy operation actually follows copy-on-write semantics, so that the actual physical copy of the bits occur only if the underlying

data in the bitmap graphics context is modified. As a consequence, you may want to use the resulting image and release it before you perform additional drawing into the bitmap graphics context. In this way, you can avoid the actual physical copy of the data.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGBitmapContext.h`

CGBitmapContextGetAlphaInfo

Returns the alpha information associated with the context, which indicates how a bitmap context handles the alpha component.

```
CGImageAlphaInfo CGBitmapContextGetAlphaInfo (
   CGContextRef c
);
```

Parameters

context

A bitmap context.

Return Value
A bitmap information constant. If the specified context is not a bitmap context, `kCGImageAlphaNone` (page 226) is returned. See `CGImageAlphaInfo` (renamed to `CGBitmapInfo` in Mac OS X v10.4) for more information about values.

Discussion
Every bitmap context contains an attribute that specifies whether the bitmap contains an alpha component, and how it is generated. The alpha component determines the opacity of a pixel when it is drawn.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGBitmapContext.h`

CGBitmapContextGetBitmapInfo

Obtains the bitmap information associated with a bitmap graphics context.

```
CGBitmapInfo CGBitmapContextGetBitmapInfo (
   CGContextRef c
);
```

Parameters

c

A bitmap graphics context.

Return Value
The bitmap info of the bitmap graphics context or `0` if `c` is not a bitmap graphics context. See *CGImage Reference* for a description of the `CGBitmapInfo` (page 227) constants that can be returned.

Discussion
The `CGBitmapInfo` data returned by the function specifies whether the bitmap contains an alpha channel and how the alpha channel is generated, along with whether the components are floating-point or integer.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGBitmapContext.h`

CGBitmapContextGetBitsPerComponent

Returns the bits per component of a bitmap context.

```
size_t CGBitmapContextGetBitsPerComponent (
   CGContextRef c
);
```

Parameters

context
The bitmap context to examine.

Return Value
The number of bits per component in the specified context, or `0` if the context is not a bitmap context.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGBitmapContext.h`

CGBitmapContextGetBitsPerPixel

Returns the bits per pixel of a bitmap context.

```
size_t CGBitmapContextGetBitsPerPixel (
   CGContextRef c
);
```

Parameters

context
The bitmap context to examine.

Return Value
The number of bits per pixel in the specified context, or `0` if the context is not a bitmap context.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGBitmapContext.h`

CGBitmapContextGetBytesPerRow

Returns the bytes per row of a bitmap context.

```
size_t CGBitmapContextGetBytesPerRow (
   CGContextRef c
);
```

Parameters

context

The bitmap context to examine.

Return Value

The number of bytes per row of the specified context, or `0` if the context is not a bitmap context.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGBitmapContext.h`

CGBitmapContextGetColorSpace

Returns the color space of a bitmap context.

```
CGColorSpaceRef CGBitmapContextGetColorSpace (
   CGContextRef c
);
```

Parameters

context

The bitmap context to examine.

Return Value

The color space of the specified context, or `NULL` if the context is not a bitmap context. You are responsible for retaining and releasing this object as necessary.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGBitmapContext.h`

CGBitmapContextGetData

Returns a pointer to the image data associated with a bitmap context.

```
void * CGBitmapContextGetData (
   CGContextRef c
);
```

Parameters

context

The bitmap context to examine.

Return Value
A pointer to the specified bitmap context's image data, or `NULL` if the context is not a bitmap context.

Availability
Available in Mac OS X version 10.2 and later.

Related Sample Code
CarbonSketch

Declared In
`CGBitmapContext.h`

CGBitmapContextGetHeight

Returns the height in pixels of a bitmap context.

```
size_t CGBitmapContextGetHeight (
   CGContextRef c
);
```

Parameters

context
The bitmap context to examine.

Return Value
The height in pixels of the specified context, or `0` if the context is not a bitmap context.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGBitmapContext.h`

CGBitmapContextGetWidth

Returns the width in pixels of a bitmap context.

```
size_t CGBitmapContextGetWidth (
   CGContextRef c
);
```

Parameters

context
The bitmap context to examine.

Return Value
The width in pixels of the specified context, or `0` if the context is not a bitmap context.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGBitmapContext.h`

CGColor Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGColor.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGColorRef` opaque type contains a set of components (such as red, green, and blue) that uniquely define a color, and a color space that specifies how those components should be interpreted. Quartz color objects provide a fast and convenient way to manage and set colors, especially colors that are used repeatedly. Quartz drawing operations use color objects for setting fill and stroke colors, managing alpha, and setting color with a pattern.

See also these related references: *CGContext Reference*, *CGColorSpace Reference*, and *CGPattern Reference*.

Functions by Task

Getting a Constant Color

`CGColorGetConstantColor` (page 38)
Returns a color object that represents a constant color.

Retaining and Releasing Color Objects

`CGColorRelease` (page 40)
Decrements the retain count of a Quartz color.

`CGColorRetain` (page 40)
Increments the retain count of a Quartz color.

Creating Quartz Colors

`CGColorCreate` (page 32)
Creates a Quartz color using a list of intensity values (including alpha) and an associated color space.

`CGColorCreateCopy` (page 33)
Creates a copy of an existing Quartz color.

`CGColorCreateGenericGray` (page 35)
Creates a color in the Generic gray color space.

`CGColorCreateGenericRGB` (page 35)
Creates a color in the Generic RGB color space.

`CGColorCreateGenericCMYK` (page 34)
Creates a color in the Generic CMYK color space.

`CGColorCreateCopyWithAlpha` (page 33)
Creates a copy of an existing Quartz color, substituting a new alpha value.

`CGColorCreateWithPattern` (page 36)
Creates a Quartz color using a list of intensity values (including alpha), a pattern color space, and a pattern.

Getting Information about Quartz Colors

`CGColorEqualToColor` (page 36)
Indicates whether two colors are equal.

`CGColorGetAlpha` (page 37)
Returns the value of the alpha component associated with a Quartz color.

`CGColorGetColorSpace` (page 37)
Returns the color space associated with a Quartz color.

`CGColorGetComponents` (page 38)
Returns the values of the color components (including alpha) associated with a Quartz color.

`CGColorGetNumberOfComponents` (page 38)
Returns the number of color components (including alpha) associated with a Quartz color.

`CGColorGetPattern` (page 39)
Returns the pattern associated with a Quartz color in a pattern color space.

`CGColorGetTypeID` (page 39)
Returns the Core Foundation type identifier for a Quartz color data type.

Functions

CGColorCreate

Creates a Quartz color using a list of intensity values (including alpha) and an associated color space.

```
CGColorRef CGColorCreate (
   CGColorSpaceRef colorspace,
   const CGFloat components[]
);
```

Parameters

colorspace

A color space for the new color. Quartz retains this object; upon return, you may safely release it.

components

An array of intensity values describing the color. The array should contain *n*+1 values that correspond to the *n* color components in the specified color space, followed by the alpha component. Each component value should be in the range appropriate for the color space. Values outside this range will be clamped to the nearest correct value.

Return Value

A new Quartz color. You are responsible for releasing this object using `CGColorRelease` (page 40).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGColor.h`

CGColorCreateCopy

Creates a copy of an existing Quartz color.

```
CGColorRef CGColorCreateCopy (
   CGColorRef color
);
```

Parameters

color

A Quartz color.

Return Value

A copy of the specified color. You are responsible for releasing this object using `CGColorRelease` (page 40).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGColor.h`

CGColorCreateCopyWithAlpha

Creates a copy of an existing Quartz color, substituting a new alpha value.

```
CGColorRef CGColorCreateCopyWithAlpha (
   CGColorRef color,
   CGFloat alpha
);
```

Parameters

color

The Quartz color to copy.

alpha

A value that specifies the desired opacity of the copy. Values outside the range `[0,1]` are clamped to `0` or `1`.

Return Value
A copy of the specified color, using the specified alpha value. You are responsible for releasing this object using `CGColorRelease` (page 40).

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGColor.h`

CGColorCreateGenericCMYK

Creates a color in the Generic CMYK color space.

```
CGColorRef CGColorCreateGenericCMYK(
    CGFloat cyan,
    CGFloat magenta,
    CGFloat yellow,
    CGFloat black,
    CGFloat alpha
);
```

Parameters

cyan

A cyan value (`0.0` - `1.0`).

magenta

A magenta value (`0.0` - `1.0`).

yellow

A yellow value (`0.0` - `1.0`).

black

A black value (`0.0` - `1.0`).

alpha

An alpha value (0.0 - 1.0).

Return Value
A color object.

Availability
Available in Mac OS X v10.5 and later.

Declared In
CGColor.h

CGColorCreateGenericGray

Creates a color in the Generic gray color space.

```
CGColorRef CGColorCreateGenericGray(
    CGFloat gray,
    CGFloat alpha
);
```

Parameters

gray
A grayscale value (0.0 - 1.0).

alpha
An alpha value (0.0 - 1.0).

Return Value
A color object.

Availability
Available in Mac OS X v10.5 and later.

Declared In
CGColor.h

CGColorCreateGenericRGB

Creates a color in the Generic RGB color space.

```
CGColorRef CGColorCreateGenericRGB(
    CGFloat red,
    CGFloat green,
    CGFloat blue,
    CGFloat alpha
);
```

Parameters

red
A red component value (0.0 - 1.0).

green
A green component value (0.0 - 1.0).

blue
A blue component value (0.0 - 1.0).

alpha
An alpha value (0.0 - 1.0).

Return Value
A color object.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
CALayerEssentials

Declared In
CGColor.h

CGColorCreateWithPattern

Creates a Quartz color using a list of intensity values (including alpha), a pattern color space, and a pattern.

```
CGColorRef CGColorCreateWithPattern (
   CGColorSpaceRef colorspace,
   CGPatternRef pattern,
   const CGFloat components[]
);
```

Parameters

colorspace
A pattern color space for the new color. Quartz retains the color space you pass in. On return, you may safely release it.

pattern
A pattern for the new color object. Quartz retains the pattern you pass in. On return, you may safely release it.

components
An array of intensity values describing the color. The array should contain `n + 1` values that correspond to the `n` color components in the specified color space, followed by the alpha component. Each component value should be in the range appropriate for the color space. Values outside this range will be clamped to the nearest correct value.

Return Value
A new Quartz color. You are responsible for releasing this object using `CGColorRelease` (page 40).

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CGColor.h

CGColorEqualToColor

Indicates whether two colors are equal.

```
bool CGColorEqualToColor (
   CGColorRef color1,
   CGColorRef color2
);
```

Parameters

color1
The first Quartz color to compare.

color2
The second Quartz color to compare.

Return Value
A Boolean value that, if `true`, indicates that the specified colors are equal. If the colors are not equal, the value is `false`.

Discussion
Two colors are equal if they have equal color spaces and numerically equal color components.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGColor.h`

CGColorGetAlpha

Returns the value of the alpha component associated with a Quartz color.

```
CGFloat CGColorGetAlpha (
   CGColorRef color
);
```

Parameters

color

A Quartz color.

Return Value
An alpha intensity value in the range `[0,1]`. The value represents the opacity of the color.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGColor.h`

CGColorGetColorSpace

Returns the color space associated with a Quartz color.

```
CGColorSpaceRef CGColorGetColorSpace (
   CGColorRef color
);
```

Parameters

color

A Quartz color.

Return Value
The Quartz color space for the specified color. You are responsible for retaining and releasing it as needed.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGColor.h`

CGColorGetComponents

Returns the values of the color components (including alpha) associated with a Quartz color.

```
const CGFloat * CGColorGetComponents (
   CGColorRef color
);
```

Parameters

color

A Quartz color.

Return Value

An array of intensity values for the color components (including alpha) associated with the specified color. The size of the array is one more than the number of components of the color space for the color.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

CGColor.h

CGColorGetConstantColor

Returns a color object that represents a constant color.

```
CGColorRef CGColorGetConstantColor(
    CFStringRef colorName
);
```

Parameters

colorName

A color name. You can pass any of the "Constant Colors" (page 41) constant.

Return Value

A color object.

Discussion

As `CGColorGetConstantColor` is not a "Copy" or "Create" function, it does not necessarily return a new reference each time it's called. As a consequence, you should not release the returned value. However, colors returned from `CGColorGetConstantColor` can be retained and released in a properly nested fashion, just as any other Core Foundation type can.

Availability

Available in Mac OS X v10.5 and later.

Declared In

CGColor.h

CGColorGetNumberOfComponents

Returns the number of color components (including alpha) associated with a Quartz color.

```
size_t CGColorGetNumberOfComponents (
   CGColorRef color
);
```

Parameters

color

A Quartz color.

Return Value

The number of color components (including alpha) associated with the specified color. This number is one more than the number of components of the color space for the color.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGColor.h`

CGColorGetPattern

Returns the pattern associated with a Quartz color in a pattern color space.

```
CGPatternRef CGColorGetPattern (
   CGColorRef color
);
```

Parameters

color

A Quartz color.

Return Value

The pattern for the specified color. You are responsible for retaining and releasing the pattern as needed.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGColor.h`

CGColorGetTypeID

Returns the Core Foundation type identifier for a Quartz color data type.

```
CFTypeID CGColorGetTypeID (
   void
);
```

Return Value

The Core Foundation type identifier for `CGColorRef`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGColor.h`

CGColorRelease

Decrements the retain count of a Quartz color.

```
void CGColorRelease (
    CGColorRef color
);
```

Parameters

color

The Quartz color to release.

Discussion

This function is equivalent to `CFRelease`, except that it does not cause an error if the `color` parameter is `NULL`.

Availability

Available in Mac OS X version 10.3 and later.

Related Sample Code

CALayerEssentials

Declared In

`CGColor.h`

CGColorRetain

Increments the retain count of a Quartz color.

```
CGColorRef CGColorRetain (
    CGColorRef color
);
```

Parameters

color

The Quartz color to retain.

Return Value

The same color you passed in as the *color* parameter.

Discussion

This function is equivalent to `CFRetain`, except that it does not cause an error if the `color` parameter is `NULL`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGColor.h`

Data Types

CGColorRef

An opaque type that represents a color used in Quartz 2D drawing.

```
typedef struct CGColor *CGColorRef;
```

Discussion
`CGColorRef` is the fundamental data type used internally by Quartz to represent colors. CGColor objects. and the functions that operate on them, provide a fast and convenient way of managing and setting colors directly, especially colors that are reused (such as black for text).

In Mac OS X version 10.3 and later, `CGColorRef` is derived from `CFTypeRef` and inherits the properties that all Core Foundation types have in common. For more information, see CFType Reference.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGColor.h`

Constants

Constant Colors

Commonly used colors.

```
const CFStringRef kCGColorWhite;
const CFStringRef kCGColorBlack;
const CFStringRef kCGColorClear;
```

Constants

`kCGColorWhite`
The white color in the Generic gray color space.
Available in Mac OS X v10.5 and later.
Declared in `CGColor.h`.

`kCGColorBlack`
The black color in the Generic gray color space.
Available in Mac OS X v10.5 and later.
Declared in `CGColor.h`.

`kCGColorClear`
The clear color in the Generic gray color space.
Available in Mac OS X v10.5 and later.
Declared in `CGColor.h`.

Declared In
`CGColor.h`

CGColorSpace Reference

Derived From:	*CFType Reference*
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGColorSpace.h
Companion guides	Quartz 2D Programming Guide CGColor Reference CGContext Reference

Overview

The `CGColorSpaceRef` opaque type encapsulates color space information that is used to specify how Quartz interprets color information. A color space specifies how color values are interpreted. A color space is multi-dimensional, and each dimension represents a specific color component. For example, the colors in an RGB color space have three dimensions or components—red, green, and blue. The intensity of each component is represented by floating point values—their range and meaning depends on the color space in question.

Different types of devices (scanners, monitors, printers) operate within different color spaces (RGB, CMYK, grayscale). Additionally, two devices of the same type (for example, color displays from different manufacturers) may operate within the same kind of color space, yet still produce a different range of colors, or gamut. Color spaces that are correctly specified ensure that an image has a consistent appearance regardless of the output device.

Quartz supports several kinds of color spaces:

- Calibrated color spaces ensure that colors appear the same when displayed on different devices. The visual appearance of the color is preserved, as far as the capabilities of the device allow.
- Device-dependent color spaces are tied to the system of color representation of a particular device. Device color spaces are not recommended when high-fidelity color preservation is important.
- Special color spaces—indexed and pattern. An indexed color space contains a color table with up to 256 entries and a base color space to which the color table entries are mapped. Each entry in the color table specifies one color in the base color space. A pattern color space is used when stroking or filling with a pattern. Pattern color spaces are supported in Mac OS X version 10.1 and later.

Functions by Task

Creating Device-Independent Color Spaces

`CGColorSpaceCreateCalibratedGray` (page 45)
Creates a calibrated grayscale color space.

`CGColorSpaceCreateCalibratedRGB` (page 46)
Creates a calibrated RGB color space.

`CGColorSpaceCreateICCBased` (page 49)
Creates a device-independent color space that is defined according to the ICC color profile specification.

`CGColorSpaceCreateLab` (page 50)
Creates a device-independent color space that is relative to human color perception, according to the CIE L*a*b* standard.

Creating Generic or Device-Dependent Color Spaces

In Mac OS X v10.4 and later, the color space returned by each of these functions is no longer device-dependent and is replaced by a generic counterpart.

`CGColorSpaceCreateDeviceCMYK` (page 47)
Creates a device-dependent CMYK color space.

`CGColorSpaceCreateDeviceGray` (page 48)
Creates a device-dependent grayscale color space.

`CGColorSpaceCreateDeviceRGB` (page 48)
Creates a device-dependent RGB color space.

`CGColorSpaceCreateWithPlatformColorSpace` (page 52)
Creates a platform-specific color space.

Creating Special Color Spaces

`CGColorSpaceCreateIndexed` (page 50)
Creates an indexed color space, consisting of colors specified by a color lookup table.

`CGColorSpaceCreatePattern` (page 51)
Creates a pattern color space.

`CGColorSpaceCreateWithName` (page 52)
Creates a specified type of Quartz color space.

Getting Information About Color Spaces

`CGColorSpaceCopyICCProfile` (page 45)
Returns a copy of the ICC profile of the provided color space.

`CGColorSpaceGetNumberOfComponents` (page 54)
Returns the number of color components in a color space.

`CGColorSpaceGetTypeID` (page 55)
Returns the Core Foundation type identifier for Quartz color spaces.

`CGColorSpaceGetModel` (page 54)
Returns the color space model of the provided color space.

`CGColorSpaceGetBaseColorSpace` (page 53)
Returns the base color space of a pattern or indexed color space.

`CGColorSpaceGetColorTableCount` (page 54)
Returns the number of entries in the color table of an indexed color space.

`CGColorSpaceGetColorTable` (page 53)
Copies the entries in the color table of an indexed color space.

Retaining and Releasing Color Spaces

`CGColorSpaceRelease` (page 55)
Decrements the retain count of a color space.

`CGColorSpaceRetain` (page 56)
Increments the retain count of a color space.

Functions

CGColorSpaceCopyICCProfile

Returns a copy of the ICC profile of the provided color space.

```
CFDataRef CGColorSpaceCopyICCProfile(
    CGColorSpaceRef space
);
```

Parameters

space
The color space whose ICC profile you want to obtain.

Return Value
The ICC profile or `NULL` if the color space does not have an ICC profile.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceCreateCalibratedGray

Creates a calibrated grayscale color space.

```
CGColorSpaceRef CGColorSpaceCreateCalibratedGray (
   const CGFloat whitePoint[3],
   const CGFloat blackPoint[3],
   CGFloat gamma
);
```

Parameters

whitePoint

An array of 3 numbers specifying the tristimulus value, in the CIE 1931 XYZ-space, of the diffuse white point.

blackPoint

An array of 3 numbers specifying the tristimulus value, in CIE 1931 XYZ-space, of the diffuse black point.

gamma

The gamma value appropriate to the imaging device.

Return Value

A new calibrated gray color space. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion

Creates a device-independent grayscale color space that represents colors relative to a reference white point. This white point is based on the whitest light that can be generated by the output device. Colors in a device-independent color space should appear the same when displayed on different devices, to the extent that the capabilities of the device allow.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGColorSpace.h`

CGColorSpaceCreateCalibratedRGB

Creates a calibrated RGB color space.

```
CGColorSpaceRef CGColorSpaceCreateCalibratedRGB (
   const CGFloat whitePoint[3],
   const CGFloat blackPoint[3],
   const CGFloat gamma[3],
   const CGFloat matrix[9]
);
```

Parameters

whitePoint

An array of 3 numbers specifying the tristimulus value, in the CIE 1931 XYZ-space, of the diffuse white point.

blackPoint

An array of 3 numbers specifying the tristimulus value, in CIE 1931 XYZ-space, of the diffuse black point.

gamma

An array of 3 numbers specifying the gamma for the red, green, and blue components of the color space.

matrix

An array of 9 numbers specifying the linear interpretation of the gamma-modified RGB values of the color space with respect to the final XYZ representation.

Return Value
A new calibrated RGB color space. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion
Creates a device-independent RGB color space that represents colors relative to a reference white point. This white point is based on the whitest light that can be generated by the output device. Colors in a device-independent color space should appear the same when displayed on different devices, to the extent that the capabilities of the device allow.

For color spaces that require a detailed gamma, such as the piecewise transfer function used in sRGB or ITU-R BT.709, you may want to use the function `CGColorSpaceCreateICCBased` (page 49) instead, because it can accurately represent these gamma curves.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceCreateDeviceCMYK

Creates a device-dependent CMYK color space.

```
CGColorSpaceRef CGColorSpaceCreateDeviceCMYK (
   void
);
```

Return Value
A device-dependent CMYK color space. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion
In Mac OS X v10.4 and later, this color space is no longer device-dependent and is replaced by the generic counterpart—`kCGColorSpaceGenericCMYK`—described in "`Color Space Names`" (page 57). If you use this function in Mac OS X v10.4 and later, colors are mapped to the generic color spaces. If you want to bypass color matching, use the color space of the destination context.

Colors in a device-dependent color space are not transformed or otherwise modified when displayed on an output device—that is, there is no attempt to maintain the visual appearance of a color. As a consequence, colors in a device color space often appear different when displayed on different output devices. For this reason, device color spaces are not recommended when color preservation is important.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceCreateDeviceGray

Creates a device-dependent grayscale color space.

```
CGColorSpaceRef CGColorSpaceCreateDeviceGray (
   void
);
```

Return Value
A device-dependent gray color space. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion
In Mac OS X v10.4 and later, this color space is no longer device-dependent and is replaced by the generic counterpart—`kCGColorSpaceGenericGray`—described in "Color Space Names" (page 57). If you use this function in Mac OS X v10.4 and later, colors are mapped to the generic color spaces. If you want to bypass color matching, use the color space of the destination context.

Colors in a device-dependent color space are not transformed or otherwise modified when displayed on an output device—that is, there is no attempt to maintain the visual appearance of a color. As a consequence, colors in a device color space often appear different when displayed on different output devices. For this reason, device color spaces are not recommended when color preservation is important.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceCreateDeviceRGB

Creates a device-dependent RGB color space.

```
CGColorSpaceRef CGColorSpaceCreateDeviceRGB (
   void
);
```

Return Value
A device-dependent RGB color space. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion
In Mac OS X v10.4 and later, this color space is no longer device-dependent and is replaced by the generic counterpart—`kCGColorSpaceGenericRGB`—described in "Color Space Names" (page 57). If you use this function in Mac OS X v10.4 and later, colors are mapped to the generic color spaces. If you want to bypass color matching, use the color space of the destination context.

Colors in a device-dependent color space are not transformed or otherwise modified when displayed on an output device—that is, there is no attempt to maintain the visual appearance of a color. As a consequence, colors in a device color space often appear different when displayed on different output devices. For this reason, device color spaces are not recommended when color preservation is important.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGColorSpace.h

CGColorSpaceCreateICCBased

Creates a device-independent color space that is defined according to the ICC color profile specification.

```
CGColorSpaceRef CGColorSpaceCreateICCBased (
   size_t nComponents,
   const CGFloat *range,
   CGDataProviderRef profile,
   CGColorSpaceRef alternate
);
```

Parameters

nComponents
The number of color components in the color space defined by the ICC profile data. This must match the number of components actually in the ICC profile and must equal 1, 3, or 4.

range
An array of numbers that specify the minimum and maximum valid values of the corresponding color components. The size of the array is two times the number of components. If `c[k]` is the `k`th color component, the valid range is range`[2*k]` ≤ `c[k]` ≤ range`[2*k+1]`.

profile
A data provider that supplies the ICC profile.

alternateSpace
An alternate color space to use in case the ICC profile is not supported. The alternate color space must have `nComponents` color components. You must supply an alternate color space. If this parameter is `NULL`, then the function returns `NULL`.

Return Value
A new ICC-based color space object. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion
This function creates an ICC-based color space from an ICC color profile, as defined by the International Color Consortium. ICC profiles define the reproducible color gamut (the range of colors supported by a device) and other characteristics of a particular output device, providing a way to accurately transform the color space of one device to the color space of another. The ICC profile is usually provided by the manufacturer of the device. Additionally, some color monitors and printers contain electronically embedded ICC profile information, as do some bitmap formats such as TIFF. Colors in a device-independent color space should appear the same when displayed on different devices, to the extent that the capabilities of the device allow.

You may want to use this function for a color space that requires a detailed gamma, such as the piecewise transfer function used in sRGB or ITU-R BT.709, because this function can accurately represent these gamma curves.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGColorSpace.h

CGColorSpaceCreateIndexed

Creates an indexed color space, consisting of colors specified by a color lookup table.

```
CGColorSpaceRef CGColorSpaceCreateIndexed (
   CGColorSpaceRef baseSpace,
   size_t lastIndex,
   const unsigned char *colorTable
);
```

Parameters

baseSpace

The color space on which the color table is based.

lastIndex

The maximum valid index value for the color table. The value must be less than or equal to 255.

colorTable

An array of `m*(lastIndex+1)` bytes, where `m` is the number of color components in the base color space. Each byte is an unsigned integer in the range `0` to `255` that is scaled to the range of the corresponding color component in the base color space.

Return Value

A new indexed color space object. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion

An indexed color space contains a color table with up to 255 entries, and a base color space to which the color table entries are mapped. Each entry in the color table specifies one color in the base color space. A value in an indexed color space is treated as an index into the color table of the color space. The data in the table is in meshed format. (For example, for an RGB color space RGB, RGB, RGB, and so on.)

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGColorSpace.h`

CGColorSpaceCreateLab

Creates a device-independent color space that is relative to human color perception, according to the CIE L*a*b* standard.

```
CGColorSpaceRef CGColorSpaceCreateLab (
   const CGFloat whitePoint[3],
   const CGFloat blackPoint[3],
   const CGFloat range[4]
);
```

Parameters

whitePoint

An array of 3 numbers that specify the tristimulus value, in the CIE 1931 XYZ-space, of the diffuse white point.

blackPoint

An array of 3 numbers that specify the tristimulus value, in CIE 1931 XYZ-space, of the diffuse black point.

range

An array of 4 numbers that specify the range of valid values for the a* and b* components of the color space. The a* component represents values running from green to red, and the b* component represents values running from blue to yellow.

Return Value
A new L*a*b* color space. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion
The CIE L*a*b* space is a nonlinear transformation of the Munsell color notation system (a system which specifies colors by hue, value, and saturation—or "chroma"—values), designed to match perceived color difference with quantitative distance in color space. The L* component represents the lightness value, the a* component represents values running from green to red, and the b* component represents values running from blue to yellow. This roughly corresponds to the way the human brain is thought to decode colors. Colors in a device-independent color space should appear the same when displayed on different devices, to the extent that the capabilities of the device allow.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceCreatePattern

Creates a pattern color space.

```
CGColorSpaceRef CGColorSpaceCreatePattern (
   CGColorSpaceRef baseSpace
);
```

Parameters

baseSpace

For masking patterns, the underlying color space that specifies the colors to be painted through the mask. For colored patterns, you should pass `NULL`.

Return Value
A new pattern color space. You are responsible for releasing this object by calling `CGColorSpaceRelease` (page 55). If unsuccessful, returns `NULL`.

Discussion
For information on creating and using patterns, see *Quartz 2D Programming Guide* and *CGPattern Reference*. Quartz retains the color space you pass in. Upon return, you may safely release it by calling `CGColorSpaceRelease` (page 55).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceCreateWithName

Creates a specified type of Quartz color space.

```
CGColorSpaceRef CGColorSpaceCreateWithName (
   CFStringRef name
);
```

Parameters

name

A color space name. See "Color Space Names" (page 57) for a list of the valid Quartz-defined names.

Return Value

A new generic color space. You are responsible for releasing this object by calling CGColorSpaceRelease (page 55). If unsuccessful, returns NULL.

Discussion

You can use this function to create a generic color space. For more information, see "Color Space Names" (page 57).

Prior to Mac OS X v10.4, you could pass this function one of the constants defined in "Named Color Spaces (Deprecated)" (page 60). As of Mac OS X v10.4, this function returns a generic color space even if you pass is one of the deprecated named color spaces.

Availability

Available in Mac OS X v10.2 and later.

Declared In

CGColorSpace.h

CGColorSpaceCreateWithPlatformColorSpace

Creates a platform-specific color space.

```
CGColorSpaceRef CGColorSpaceCreateWithPlatformColorSpace (
   void *platformColorSpaceReference
);
```

Parameters

platformColorSpace

A generic pointer to a platform-specific color space. In Mac OS X, pass a CMProfileRef—a ColorSync profile. Quartz uses this pointer (and the underlying information) only during the function call.

Return Value

A new color space. You are responsible for releasing this object by calling CGColorSpaceRelease (page 55). If unsuccessful, returns NULL.

Discussion

Colors in a device-dependent color space are not transformed or otherwise modified when displayed on an output device—that is, there is no attempt to maintain the visual appearance of a color. As a consequence, colors in a device color space often appear different when displayed on different output devices. For this reason, device color spaces are not recommended when color preservation is important.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code
CarbonSketch

Declared In
`CGColorSpace.h`

CGColorSpaceGetBaseColorSpace

Returns the base color space of a pattern or indexed color space.

```
CGColorSpace CGColorSpaceGetBaseColorSpace(
    CGColorSpaceRef space
);
```

Parameters

space

A color space object for a pattern or indexed color space.

Return Value
The base color space if the `space` parameter is a pattern or indexed color space; otherwise, `NULL`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceGetColorTable

Copies the entries in the color table of an indexed color space.

```
void CGColorSpaceGetColorTable(
        CGColorSpaceRef space,
        unsigned char *table);
);
```

Parameters

space

A color space object for an indexed color space.

table

The array pointed to by table should be at least as large as the number of entries in the color table. On output, the array contains the table data in the same format as that passed to `CGColorSpaceCreateIndexed` (page 50).

Discussion
This function does nothing if the color space is not an indexed color space. To determine whether a color space is an indexed color space, call the function `CGColorSpaceGetModel` (page 54).

Availability
Available in Mac OS X v10.5 and later.

See Also
`CGColorSpaceGetColorTableCount` (page 54)

Declared In
CGColorSpace.h

CGColorSpaceGetColorTableCount

Returns the number of entries in the color table of an indexed color space.

```
size_t CGColorSpaceGetColorTableCount(
    CGColorSpaceRef space
);
```

Parameters

space
A color space object for an indexed color space.

Return Value
The number of entries in the color table of the `space` parameter if the color space is an indexed color space; otherwise, returns `0`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`CGColorSpaceGetColorTable` (page 53)

Declared In
CGColorSpace.h

CGColorSpaceGetModel

Returns the color space model of the provided color space.

```
CGColorSpaceModel CGColorSpaceGetModel(
    CGColorSpaceRef space
);
```

Parameters

space
A color space object.

Return Value
One of the constants described in "Color Space Models" (page 57).

Availability
Available in Mac OS X v10.5 and later.

Declared In
CGColorSpace.h

CGColorSpaceGetNumberOfComponents

Returns the number of color components in a color space.

```
size_t CGColorSpaceGetNumberOfComponents (
   CGColorSpaceRef cs
);
```

Parameters

cs

The Quartz color space to examine.

Return Value

The number of color components in the specified color space, not including the alpha value. For example, for an RGB color space, `CGColorSpaceGetNumberOfComponents` returns a value of `3`.

Discussion

A color space defines an n-dimensional space whose dimensions (or components) represent intensity values. For example, you specify colors in RGB space as three intensity values: red, green, and blue. You can use the `CGColorSpaceGetNumberOfComponents` function to obtain the number of components in a given color space.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGColorSpace.h`

CGColorSpaceGetTypeID

Returns the Core Foundation type identifier for Quartz color spaces.

```
CFTypeID CGColorSpaceGetTypeID (
   void
);
```

Return Value

The identifier for the opaque type `CGColorSpaceRef` (page 56).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGColorSpace.h`

CGColorSpaceRelease

Decrements the retain count of a color space.

```
void CGColorSpaceRelease (
   CGColorSpaceRef cs
);
```

Parameters

cs

The Quartz color space to release.

Discussion

This function is equivalent to `CFRelease`, except that it does not cause an error if the `cs` parameter is `NULL`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGColorSpace.h`

CGColorSpaceRetain

Increments the retain count of a color space.

```
CGColorSpaceRef CGColorSpaceRetain (
   CGColorSpaceRef cs
);
```

Parameters

cs

The Quartz color space to retain.

Return Value
The same color space you passed in as the *cs* parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `cs` parameter is `NULL`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGColorSpace.h`

Data Types

CGColorSpaceRef

An opaque type that encapsulates color space information.

```
typedef struct CGColorSpace *CGColorSpaceRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGColorSpace.h`

Constants

Color Space Names

Convenience constants for commonly used color spaces.

```
CFStringRef kCGColorSpaceGenericGray
CFStringRef kCGColorSpaceGenericRGB
CFStringRef kCGColorSpaceGenericCMYK
CFStringRef kCGColorSpaceGenericRGBLinear
CFStringRef kCGColorSpaceAdobeRGB1998
CFStringRef kCGColorSpaceSRGB
```

Constants

`kCGColorSpaceGenericGray`
The name of the generic gray color space.

`kCGColorSpaceGenericRGB`
The name of the generic RGB color space.

`kCGColorSpaceGenericCMYK`
The name of the generic CMYK color space.

`kCGColorSpaceGenericRGBLinear`
The name of the generic linear RGB color space. This is the same as `kCGColorSpaceGenericRGB` (page 57), but with a gamma equal to `1.0`.

`kCGColorSpaceAdobeRGB1998`
The name of the Adobe RGB (1998) color space. For more information, see "Adobe RGB (1998) Color Image Encoding", Version 2005-05, Adobe Systems Inc. (http://www.adobe.com).

`kCGColorSpaceSRGB`
The name of the SRGB color space.

Discussion

A color space name constant can be passed as a parameter to the function `CGColorSpaceCreateWithName` (page 52). These color spaces replace `"Named Color Spaces (Deprecated)"` (page 60), which are deprecated in Mac OS X v10.4.

Declared In

`CGColorSpace.h`

Color Space Models

Models for color spaces.

```
enum CGColorSpaceModel {
    kCGColorSpaceModelUnknown = -1,
    kCGColorSpaceModelMonochrome,
    kCGColorSpaceModelRGB,
    kCGColorSpaceModelCMYK,
    kCGColorSpaceModelLab,
    kCGColorSpaceModelDeviceN,
    kCGColorSpaceModelIndexed,
    kCGColorSpaceModelPattern
};
typedef int32_t CGColorSpaceModel;
```

Constants

`kCGColorSpaceModelUnknown`
An unknown color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

`kCGColorSpaceModelMonochrome`
A monochrome color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

`kCGColorSpaceModelRGB`
An RGB color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

`kCGColorSpaceModelCMYK`
A CMYK color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

`kCGColorSpaceModelLab`
A Lab color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

`kCGColorSpaceModelDeviceN`
A DeviceN color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

`kCGColorSpaceModelIndexed`
An indexed color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

`kCGColorSpaceModelPattern`
A pattern color space model.
Available in Mac OS X v10.5 and later.
Declared in `CGColorSpace.h`.

Declared In
`CGColorSpace.h`

Color Rendering Intents

Handling options for colors that are not located within the destination color space of a graphics context.

```
enum CGColorRenderingIntent {
    kCGRenderingIntentDefault,
    kCGRenderingIntentAbsoluteColorimetric,
    kCGRenderingIntentRelativeColorimetric,
    kCGRenderingIntentPerceptual,
    kCGRenderingIntentSaturation
};
typedef enum CGColorRenderingIntent CGColorRenderingIntent;
```

Constants

`kCGRenderingIntentDefault`
The default rendering intent for the graphics context.

Available in Mac OS X v10.0 and later.

Declared in `CGColorSpace.h`.

`kCGRenderingIntentAbsoluteColorimetric`
Map colors outside of the gamut of the output device to the closest possible match inside the gamut of the output device. This can produce a clipping effect, where two different color values in the gamut of the graphics context are mapped to the same color value in the output device's gamut. Unlike the relative colorimetric, absolute colorimetric does not modify colors inside the gamut of the output device.

Available in Mac OS X v10.0 and later.

Declared in `CGColorSpace.h`.

`kCGRenderingIntentRelativeColorimetric`
Map colors outside of the gamut of the output device to the closest possible match inside the gamut of the output device. This can produce a clipping effect, where two different color values in the gamut of the graphics context are mapped to the same color value in the output device's gamut. The relative colorimetric shifts all colors (including those within the gamut) to account for the difference between the white point of the graphics context and the white point of the output device.

Available in Mac OS X v10.0 and later.

Declared in `CGColorSpace.h`.

`kCGRenderingIntentPerceptual`
Preserve the visual relationship between colors by compressing the gamut of the graphics context to fit inside the gamut of the output device. Perceptual intent is good for photographs and other complex, detailed images.

Available in Mac OS X v10.0 and later.

Declared in `CGColorSpace.h`.

`kCGRenderingIntentSaturation`
Preserve the relative saturation value of the colors when converting into the gamut of the output device. The result is an image with bright, saturated colors. Saturation intent is good for reproducing images with low detail, such as presentation charts and graphs.

Available in Mac OS X v10.0 and later.

Declared in `CGColorSpace.h`.

Discussion
The rendering intent specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context. It determines the exact method used to map colors from one color space to another. If you do not explicitly set the rendering intent by calling the function `CGContextSetRenderingIntent` (page 120), the graphics context uses the relative colorimetric rendering intent, except when drawing sampled images.

Declared In
`CGColorSpace.h`

Named Color Spaces (Deprecated)

Color spaces used in the Preferences application.

```
#define kCGColorSpaceUserCMYK CFSTR("kCGColorSpaceUserCMYK")
#define kCGColorSpaceUserGray CFSTR("kCGColorSpaceUserGray")
#define kCGColorSpaceUserRGB CFSTR("kCGColorSpaceUserRGB")
```

Constants

`kCGColorSpaceUserCMYK`
A user-defined CMYK color space.

`kCGColorSpaceUserGray`
A user-defined gray color space.

`kCGColorSpaceUserRGB`
A user-defined RGB color space.

Discussion
These constants are deprecated in Mac OS X v10.4. Instead use "`Color Space Names`" (page 57).

The named color spaces are user-configurable in the "Default Profiles for Documents" pane, located in Mac OS 10.2 in the ColorSync preference panel, and in Mac OS 10.3 in the Displays Color Preference panel. See also `CGColorSpaceCreateWithName` (page 52).

Availability
Available in Mac OS X v10.2 and later but deprecated in Mac OS X v10.4.

Declared In
`CGColorSpace.h`

CGContext Reference

Derived From:	*CFType Reference*
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGContext.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGContextRef` opaque type represents a Quartz 2D drawing destination. A graphics context contains drawing parameters and all device-specific information needed to render the paint on a page to the destination, whether the destination is a window in an application, a bitmap image, a PDF document, or a printer. You can obtain a graphics context by using Quartz graphics context creation functions or by using higher-level functions provided in the Carbon, Cocoa, or Printing frameworks. Quartz provides creation functions for various flavors of Quartz graphics contexts including bitmap images and PDF. The Carbon and Cocoa frameworks provide functions for obtaining window graphics contexts. The Printing framework provides functions that obtain a graphics context appropriate for the destination printer.

Functions by Task

Managing Graphics Contexts

`CGContextFlush` (page 95)
Forces all pending drawing operations in a window context to be rendered immediately to the destination device.

`CGContextGetTypeID` (page 99)
Returns the type identifier for Quartz graphics contexts.

`CGContextRelease` (page 102)
Decrements the retain count of a graphics context.

`CGContextRetain` (page 103)
Increments the retain count of a graphics context.

`CGContextSynchronize` (page 136)
Marks a window context for update.

Saving and Restoring the Current Graphics State

`CGContextSaveGState` (page 104)
Pushes a copy of the current graphics state onto the graphics state stack for the context.

`CGContextRestoreGState` (page 103)
Sets the current graphics state to the state most recently saved.

Getting and Setting Graphics State Parameters

`CGContextGetInterpolationQuality` (page 97)
Returns the current level of interpolation quality for a graphics context.

`CGContextSetFlatness` (page 113)
Sets the accuracy of curved paths in a graphics context.

`CGContextSetInterpolationQuality` (page 116)
Sets the level of interpolation quality for a graphics context.

`CGContextSetLineCap` (page 116)
Sets the style for the endpoints of lines drawn in a graphics context.

`CGContextSetLineDash` (page 117)
Sets the pattern for dashed lines in a graphics context.

`CGContextSetLineJoin` (page 118)
Sets the style for the joins of connected lines in a graphics context.

`CGContextSetLineWidth` (page 118)
Sets the line width for a graphics context.

`CGContextSetMiterLimit` (page 119)
Sets the miter limit for the joins of connected lines in a graphics context.

`CGContextSetPatternPhase` (page 119)
Sets the pattern phase of a context.

`CGContextSetFillPattern` (page 112)
Sets the fill pattern in the specified graphics context.

`CGContextSetRenderingIntent` (page 120)
Sets the rendering intent in the current graphics state.

`CGContextSetShouldAntialias` (page 124)
Sets anti-aliasing on or off for a graphics context.

`CGContextSetShouldSmoothFonts` (page 124)
Enables or disables font smoothing in a graphics context.

`CGContextSetStrokePattern` (page 126)
Sets the stroke pattern in the specified graphics context.

`CGContextSetBlendMode` (page 107)
Sets how Quartz composites sample values for a graphics context.

`CGContextSetAllowsAntialiasing` (page 106)
Sets whether or not to allow anti-aliasing for a graphics context.

Constructing Paths

These functions are used to define the geometry of the current path.

`CGContextAddArc` (page 68)
Adds an arc of a circle to the current path, using a center point, radius, and end point.

`CGContextAddArcToPoint` (page 69)
Adds an arc of a circle to the current path, using a radius and tangent points.

`CGContextAddCurveToPoint` (page 70)
Appends a cubic Bézier curve from the current point, using the provided control points and end point .

`CGContextAddLines` (page 72)
Adds a sequence of connected straight-line segments to the current path.

`CGContextAddLineToPoint` (page 73)
Appends a straight line segment from the current point to the provided point .

`CGContextAddPath` (page 73)
Adds a previously created Quartz path object to the current path in a graphics context.

`CGContextAddQuadCurveToPoint` (page 74)
Appends a quadratic Bézier curve from the current point, using a control point and an end point you specify.

`CGContextAddRect` (page 75)
Adds a rectangular path to the current path.

`CGContextAddRects` (page 75)
Adds a set rectangular paths to the current path.

`CGContextBeginPath` (page 76)
Creates a new empty path in a graphics context.

`CGContextClosePath` (page 81)
Closes and terminates an open path.

`CGContextMoveToPoint` (page 100)
Begins a new path at the point you specify.

`CGContextAddEllipseInRect` (page 71)
Adds an ellipse that fits inside the specified rectangle.

Painting Paths

These functions are used to stroke along or fill in the current path.

`CGContextClearRect` (page 78)
Paints a transparent rectangle.

`CGContextDrawPath` (page 87)
Draws the current path using the provided drawing mode.

`CGContextEOFillPath` (page 93)
Paints the area within the current path, using the even-odd fill rule.

`CGContextFillPath` (page 94)
Paints the area within the current path, using the nonzero winding number rule.

`CGContextFillRect` (page 94)
Paints the area contained within the provided rectangle, using the fill color in the current graphics state.

`CGContextFillRects` (page 95)
Paints the areas contained within the provided rectangles, using the fill color in the current graphics state.

`CGContextFillEllipseInRect` (page 93)
Paints the area of the ellipse that fits inside the provided rectangle, using the fill color in the current graphics state.

`CGContextStrokePath` (page 134)
Paints a line along the current path.

`CGContextStrokeRect` (page 134)
Paints a rectangular path.

`CGContextStrokeRectWithWidth` (page 135)
Paints a rectangular path, using the specified line width.

`CGContextReplacePathWithStrokedPath` (page 102)
Replaces the path in the graphics context with the stroked version of the path.

`CGContextStrokeEllipseInRect` (page 133)
Strokes an ellipse that fits inside the specified rectangle.

`CGContextStrokeLineSegments` (page 133)
Strokes a sequence of line segments.

Getting Information About Paths

`CGContextIsPathEmpty` (page 100)
Indicates whether the current path contains any subpaths.

`CGContextGetPathCurrentPoint` (page 98)
Returns the current point in a non-empty path.

`CGContextGetPathBoundingBox` (page 97)
Returns the smallest rectangle that contains the current path.

`CGContextPathContainsPoint` (page 101)
Checks to see whether the specified point is contained in the current path.

Modifying Clipping Paths

`CGContextClip` (page 79)
Modifies the current clipping path, using the nonzero winding number rule.

`CGContextEOClip` (page 92)
Modifies the current clipping path, using the even-odd rule.

`CGContextClipToRect` (page 80)
Sets the clipping path to the intersection of the current clipping path with the area defined by the specified rectangle.

`CGContextClipToRects` (page 81)
Sets the clipping path to the intersection of the current clipping path with the region defined by an array of rectangles.

`CGContextGetClipBoundingBox` (page 96)
Returns the bounding box of a clipping path.

`CGContextClipToMask` (page 79)
Maps a mask into the specified rectangle and intersects it with the current clipping area of the graphics context.

Setting Color, Color Space, and Shadow Values

`CGContextSetAlpha` (page 107)
Sets the opacity level for objects drawn in a graphics context.

`CGContextSetCMYKFillColor` (page 108)
Sets the current fill color to a value in the DeviceCMYK color space.

`CGContextSetFillColor` (page 111)
Sets the current fill color.

`CGContextSetCMYKStrokeColor` (page 110)
Sets the current stroke color to a value in the DeviceCMYK color space.

`CGContextSetFillColorSpace` (page 111)
Sets the fill color space in a graphics context.

`CGContextSetFillColorWithColor` (page 112)
Sets the current fill color in a graphics context, using a Quartz color.

`CGContextSetGrayFillColor` (page 114)
Sets the current fill color to a value in the DeviceGray color space.

`CGContextSetGrayStrokeColor` (page 115)
Sets the current stroke color to a value in the DeviceGray color space.

`CGContextSetRGBFillColor` (page 120)
Sets the current fill color to a value in the DeviceRGB color space.

`CGContextSetRGBStrokeColor` (page 121)
Sets the current stroke color to a value in the DeviceRGB color space.

`CGContextSetShadow` (page 122)
Enables shadowing in a graphics context.

`CGContextSetShadowWithColor` (page 123)
Enables shadowing with color a graphics context.

`CGContextSetStrokeColor` (page 125)
Sets the current stroke color.

`CGContextSetStrokeColorSpace` (page 125)
Sets the stroke color space in a graphics context.

`CGContextSetStrokeColorWithColor` (page 126)
Sets the current stroke color in a context, using a Quartz color.

Transforming User Space

These functions allow you to examine and change the current transformation matrix (CTM) in a graphics context.

`CGContextConcatCTM` (page 82)
: Transforms the user coordinate system in a context using a specified matrix.

`CGContextGetCTM` (page 96)
: Returns the current transformation matrix.

`CGContextRotateCTM` (page 104)
: Rotates the user coordinate system in a context.

`CGContextScaleCTM` (page 105)
: Changes the scale of the user coordinate system in a context.

`CGContextTranslateCTM` (page 136)
: Changes the origin of the user coordinate system in a context.

Using Transparency Layers

`CGContextBeginTransparencyLayer` (page 77)
: Begins a transparency layer.

`CGContextBeginTransparencyLayerWithRect` (page 78)
: Begins a transparency layer whose contents are bounded by the specified rectangle.

`CGContextEndTransparencyLayer` (page 92)
: Ends a transparency layer.

Drawing an Image to a Graphics Context

`CGContextDrawTiledImage` (page 90)
: Repeatedly draws an image, scaled to the provided rectangle, to fill the current clip region.

`CGContextDrawImage` (page 86)
: Draws an image into a graphics context.

Drawing PDF Content to a Graphics Context

`CGContextDrawPDFDocument` (page 88)
: Draws a page of a PDF document into a graphics context.

`CGContextDrawPDFPage` (page 88)
: Draws a page in the current user space of a PDF context.

Drawing With a Gradient

`CGContextDrawLinearGradient` (page 86)
: Paints a gradient fill that varies along the line defined by the provided starting and ending points.

`CGContextDrawRadialGradient` (page 89)
Paints a gradient fill that varies along the area defined by the provided starting and ending circles.

Drawing With a Shading

`CGContextDrawShading` (page 90)
Fills the clipping path of a context with the specified shading.

Setting Up a Page-Based Graphics Context

`CGContextBeginPage` (page 76)
Starts a new page in a page-based graphics context.

`CGContextEndPage` (page 91)
Ends the current page in a page-based graphics context.

Drawing Glyphs

`CGContextShowGlyphs` (page 129)
Displays an array of glyphs at the current text position.

`CGContextShowGlyphsAtPoint` (page 129)
Displays an array of glyphs at a position you specify.

`CGContextShowGlyphsWithAdvances` (page 130)
Draws an array of glyphs with varying offsets.

`CGContextShowGlyphsAtPositions` (page 130)
Draws glyphs at the provided position.

Drawing Text

`CGContextGetTextMatrix` (page 98)
Returns the current text matrix.

`CGContextGetTextPosition` (page 99)
Returns the location at which text is drawn.

`CGContextSelectFont` (page 106)
Sets the font and font size in a graphics context.

`CGContextSetCharacterSpacing` (page 108)
Sets the current character spacing.

`CGContextSetFont` (page 113)
Sets the platform font in a graphics context.

`CGContextSetFontSize` (page 114)
Sets the current font size.

`CGContextSetTextDrawingMode` (page 127)
Sets the current text drawing mode.

`CGContextSetTextMatrix` (page 127)
Sets the current text matrix.

`CGContextSetTextPosition` (page 128)
Sets the location at which text is drawn.

`CGContextShowText` (page 131)
Displays a character array at the current text position, a point specified by the current text matrix.

`CGContextShowTextAtPoint` (page 132)
Displays a character string at a position you specify.

Converting Between Device Space and User Space

`CGContextGetUserSpaceToDeviceSpaceTransform` (page 100)
Returns an affine transform that maps user space coordinates to device space coordinates.

`CGContextConvertPointToDeviceSpace` (page 83)
Returns a point that is transformed from user space coordinates to device space coordinates.

`CGContextConvertPointToUserSpace` (page 83)
Returns a point that is transformed from device space coordinates to user space coordinates.

`CGContextConvertSizeToDeviceSpace` (page 85)
Returns a size that is transformed from user space coordinates to device space coordinates.

`CGContextConvertSizeToUserSpace` (page 85)
Returns a size that is transformed from device space coordinates to user space coordinates

`CGContextConvertRectToDeviceSpace` (page 84)
Returns a rectangle that is transformed from user space coordinate to device space coordinates.

`CGContextConvertRectToUserSpace` (page 84)
Returns a rectangle that is transformed from device space coordinate to user space coordinates.

Functions

CGContextAddArc

Adds an arc of a circle to the current path, using a center point, radius, and end point.

```
void CGContextAddArc (
   CGContextRef c,
   CGFloat x,
   CGFloat y,
   CGFloat radius,
   CGFloat startAngle,
   CGFloat endAngle,
   int clockwise
);
```

Parameters

context
A graphics context.

x
The x-value, in user space coordinates, for the center of the arc.

y
The y-value, in user space coordinates, for the center of the arc.

radius
The radius of the arc, in user space coordinates.

startAngle
The angle to the starting point of the arc, measured in radians from the positive x-axis.

endAngle
The angle to the end point of the arc, measured in radians from the positive x-axis.

clockwise
Pass `1` to draw the arc clockwise; `0` otherwise.

Discussion
When you call this function, Quartz builds an arc of a circle centered on the point you provide. The arc is of the specified radius and extends between the start and end point. (You can also use `CGContextAddArc` as a convenient way to draw a circle, by setting the start point to `0` and the end point to `2*Pi`.)

If the current path already contains a subpath, Quartz additionally appends a straight line segment from the current point to the starting point of the arc. If the current path is empty, Quartz creates a new subpath for the arc and does not add the initial straight line segment.

After adding the arc, the current point is reset to the end point of arc (the second tangent point).

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextAddArcToPoint` (page 69)

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextAddArcToPoint

Adds an arc of a circle to the current path, using a radius and tangent points.

```
void CGContextAddArcToPoint (
   CGContextRef c,
   CGFloat x1,
   CGFloat y1,
   CGFloat x2,
   CGFloat y2,
   CGFloat radius
);
```

Parameters

context
A graphics context whose current path is not empty.

x1

The x-value, in user space coordinates, for the end point of the first tangent line. The first tangent line is drawn from the current point to (x1,y1).

y1

The y-value, in user space coordinates, for the end point of the first tangent line. The first tangent line is drawn from the current point to (x1,y1).

x2

The x-value, in user space coordinates, for the end point of the second tangent line. The second tangent line is drawn from (x1,y1) to (x2,y2).

y2

The y-value, in user space coordinates, for the end point of the second tangent line. The second tangent line is drawn from (x1,y1) to (x2,y2).

radius

The radius of the arc, in user space coordinates.

Discussion

This function draws an arc that is tangent to the line from the current point to `(x1,y1)` and to the line from `(x1,y1)` to *(x2,y2)*. The start and end points of the arc are located on the first and second tangent lines, respectively. The start and end points of the arc are also the "tangent points" of the lines.

If the current point and the first tangent point of the arc (the starting point) are not equal, Quartz appends a straight line segment from the current point to the first tangent point. After adding the arc, the current point is reset to the end point of arc (the second tangent point).

Availability

Available in Mac OS X version 10.0 and later.

See Also

`CGContextAddArc` (page 68)

`CGContextAddArcToPoint` (page 69)

Related Sample Code

CarbonSketch

Declared In

`CGContext.h`

CGContextAddCurveToPoint

Appends a cubic Bézier curve from the current point, using the provided control points and end point .

```
void CGContextAddCurveToPoint (
   CGContextRef c,
   CGFloat cp1x,
   CGFloat cp1y,
   CGFloat cp2x,
   CGFloat cp2y,
   CGFloat x,
   CGFloat y
);
```

Parameters

context

A graphics context whose current path is not empty.

cp1x

The x-value, in user space coordinates, for the first control point of the curve.

cp1y

The y-value, in user space coordinates, for the first control point of the curve.

cp2x

The x-value, in user space coordinates, for the second control point of the curve.

cp2y

The y-value, in user space coordinates, for the second control point of the curve.

x

The x-value, in user space coordinates, at which to end the curve.

y

The y-value, in user space coordinates, at which to end the curve.

Discussion

This function appends a cubic curve to the current path. After adding the segment, the current point is reset from the beginning of the new segment to the end point of that segment.

Availability

Available in Mac OS X version 10.0 and later.

See Also

`CGContextAddQuadCurveToPoint` (page 74)

`CGContextAddArcToPoint` (page 69)

Declared In

`CGContext.h`

CGContextAddEllipseInRect

Adds an ellipse that fits inside the specified rectangle.

```
void CGContextAddEllipseInRect (
   CGContextRef context,
   CGRect rect
);
```

Parameters

context

A graphics context.

rect

A rectangle that defines the area for the ellipse to fit in.

Discussion
The ellipse is approximated by a sequence of Bézier curves. Its center is the midpoint of the rectangle defined by the `rect` parameter. If the rectangle is square, then the ellipse is circular with a radius equal to one-half the width (or height) of the rectangle. If the `rect` parameter specifies a rectangular shape, then the major and minor axes of the ellipse are defined by the width and height of the rectangle.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGContext.h`

CGContextAddLines

Adds a sequence of connected straight-line segments to the current path.

```
void CGContextAddLines (
   CGContextRef c,
   const CGPoint points[],
   size_t count
);
```

Parameters

context

A graphics context .

points

An array of values that specify the start and end points of the line segments to draw. Each point in the array specifies a position in user space. The first point is the array specifies the initial starting point.

count

The number of elements in the `points` array.

Discussion
This is a convenience function that adds a sequence of connected line segments to the current path in a graphics context. Quartz connects each point in the array with the subsequent point in the array, using straight line segments.

On return, the current point is the last point in the array. This function does not automatically close the path created by the line segments. If you want to close the path, you must call `CGContextClosePath` (page 81).

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextAddLineToPoint` (page 73)

Declared In
`CGContext.h`

CGContextAddLineToPoint

Appends a straight line segment from the current point to the provided point .

```
void CGContextAddLineToPoint (
   CGContextRef c,
   CGFloat x,
   CGFloat y
);
```

Parameters

context

A graphics context whose current path is not empty.

x

The x-value, in user space coordinates, for the end of the line segment.

y

The y-value, in user space coordinates, for the end of the line segment.

Discussion

After adding the line segment, the current point is reset from the beginning of the new line segment to the endpoint of that line segment.

Availability

Available in Mac OS X version 10.0 and later.

See Also

`CGContextAddLines` (page 72)

Related Sample Code

CALayerEssentials

CarbonSketch

HID Calibrator

HID Explorer

Declared In

`CGContext.h`

CGContextAddPath

Adds a previously created Quartz path object to the current path in a graphics context.

```
void CGContextAddPath (
   CGContextRef context,
   CGPathRef path
);
```

Parameters

context

A graphics context .

path

A previously created Quartz path object. See *CGPath Reference*.

Discussion
Quartz applies the current transformation matrix (CTM) to the points in the new path before they are added to the current path in the graphics context.

Availability
Available in Mac OS X version 10.2 and later.

Related Sample Code
CALayerEssentials

Declared In
`CGContext.h`

CGContextAddQuadCurveToPoint

Appends a quadratic Bézier curve from the current point, using a control point and an end point you specify.

```
void CGContextAddQuadCurveToPoint (
   CGContextRef c,
   CGFloat cpx,
   CGFloat cpy,
   CGFloat x,
   CGFloat y
);
```

Parameters

context
A graphics context whose current path is not empty.

cpx
The x-coordinate of the user space for the control point of the curve.

cpy
The y-coordinate of the user space for the control point of the curve.

x
The x-coordinate of the user space at which to end the curve.

y
The y-coordinate of the user space at which to end the curve.

Discussion
This function appends a quadratic curve to the current subpath. After adding the segment, the current point is reset from the beginning of the new segment to the end point of that segment.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextAddCurveToPoint` (page 70)
`CGContextAddArcToPoint` (page 69)

Declared In
`CGContext.h`

CGContextAddRect

Adds a rectangular path to the current path.

```
void CGContextAddRect (
   CGContextRef c,
   CGRect rect
);
```

Parameters

context

A graphics context.

rect

A rectangle, specified in user space coordinates.

Availability

Available in Mac OS X version 10.0 and later.

See Also

CGContextAddRects (page 75)

Related Sample Code

CarbonSketch

Declared In

CGContext.h

CGContextAddRects

Adds a set rectangular paths to the current path.

```
void CGContextAddRects (
   CGContextRef c,
   const CGRect rects[],
   size_t count
);
```

Parameters

context

A graphics context.

rects

An array of rectangles, specified in user space coordinates.

count

The number of rectangles in the `rects` array.

Availability

Available in Mac OS X version 10.0 and later.

See Also

CGContextAddRect (page 75)

Declared In

CGContext.h

CGContextBeginPage

Starts a new page in a page-based graphics context.

```
void CGContextBeginPage (
   CGContextRef c,
   const CGRect *mediaBox
);
```

Parameters

context

A page-based graphics context such as a PDF context. If you specify a context that does not support multiple pages, this function does nothing.

mediaBox

A Quartz rectangle defining the bounds of the new page, expressed in units of the default user space, or `NULL`. These bounds supersede any supplied for the media box when you created the context. If you pass `NULL`, Quartz uses the rectangle you supplied for the media box when the graphics context was created.

Discussion

When using a graphics context that supports multiple pages, you should call this function together with `CGContextEndPage` (page 91) to delineate the page boundaries in the output. In other words, each page should be bracketed by calls to `CGContextBeginPage` and `CGContextEndPage`. Quartz ignores all drawing operations performed outside a page boundary in a page-based context.

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CGContext.h`

CGContextBeginPath

Creates a new empty path in a graphics context.

```
void CGContextBeginPath (
   CGContextRef c
);
```

Parameters

context

A graphics context.

Discussion

A graphics context can have only a single path in use at any time. If the specified context already contains a current path when you call this function, Quartz replaces the previous current path with the new path. In this case, Quartz discards the old path and any data associated with it.

The current path is not part of the graphics state. Consequently, saving and restoring the graphics state has no effect on the current path.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextClosePath` (page 81)

Related Sample Code
CarbonSketch
HID Calibrator
HID Explorer

Declared In
`CGContext.h`

CGContextBeginTransparencyLayer

Begins a transparency layer.

```
void CGContextBeginTransparencyLayer (
   CGContextRef context,
   CFDictionaryRef auxiliaryInfo
);
```

Parameters

context
A graphics context.

auxiliaryInfo
A dictionary that specifies any additional information, or `NULL`.

Discussion
Until a corresponding call to `CGContextEndTransparencyLayer` (page 92), all subsequent drawing operations in the specified context are composited into a fully transparent backdrop (which is treated as a separate destination buffer from the context).

After a call to `CGContextEndTransparencyLayer`, the result is composited into the context using the global alpha and shadow state of the context. This operation respects the clipping region of the context.

After a call to this function, all of the parameters in the graphics state remain unchanged with the exception of the following:

- The global alpha is set to `1`.
- The shadow is turned off.

Ending the transparency layer restores these parameters to their previous values. Quartz maintains a transparency layer stack for each context, and transparency layers may be nested.

Tip: For best performance, make sure that you set the smallest possible clipping area for the objects in the transparency layer prior to calling `CGContextBeginTransparencyLayer`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CGContext.h

CGContextBeginTransparencyLayerWithRect

Begins a transparency layer whose contents are bounded by the specified rectangle.

```
void CGContextBeginTransparencyLayerWithRect(CGContextRef context, CGRect rect,
CFDictionaryRef auxiliaryInfo);
```

Parameters

context
A graphics context.

rect
The rectangle, specified in user space, that bounds the transparency layer.

auxiliaryInfo
A dictionary that specifies any additional information, or NULL.

Discussion
This function is identical to CGContextBeginTransparencyLayer (page 77) except that the content of the transparency layer is within the bounds of the provided rectangle.

Availability
Available in Mac OS X v10.5 and later.

Declared In
CGContext.h

CGContextClearRect

Paints a transparent rectangle.

```
void CGContextClearRect (
   CGContextRef c,
   CGRect rect
);
```

Parameters

context
The graphics context in which to paint the rectangle.

rect
The rectangle, in user space coordinates.

Discussion
If the provided context is a window or bitmap context, Quartz effectively clears the rectangle. For other context types, Quartz fills the rectangle in a device-dependent manner. However, you should not use this function in contexts other than window or bitmap contexts.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextClip

Modifies the current clipping path, using the nonzero winding number rule.

```
void CGContextClip (
   CGContextRef c
);
```

Parameters

context

A graphics context that contains a path. If the context does not have a current path, the function does nothing.

Discussion
The function uses the nonzero winding number rule to calculate the intersection of the current path with the current clipping path. Quartz then uses the path resulting from the intersection as the new current clipping path for subsequent painting operations.

Unlike the current path, the current clipping path is part of the graphics state. Therefore, to re-enlarge the paintable area by restoring the clipping path to a prior state, you must save the graphics state before you clip and restore the graphics state after you've completed any clipped drawing.

After determining the new clipping path, the function resets the context's current path to an empty path.

See also`CGContextEOClip` (page 92)

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGContext.h`

CGContextClipToMask

Maps a mask into the specified rectangle and intersects it with the current clipping area of the graphics context.

```
void CGContextClipToMask (
   CGContextRef c,
   CGRect rect,
   CGImageRef mask
);
```

Parameters

c

A graphics context.

rect

The rectangle to map the `mask` parameter to.

mask

An image or an image mask. If `mask` is an image, then it must be in the DeviceGray color space, may not have an alpha component, and may not be masked by an image mask or masking color.

Discussion
If the *mask* parameter is an image mask, then Quartz clips in a manner identical to the behavior seen with the function `CGContextDrawImage`—the mask indicates an area to be left unchanged when drawing. The source samples of the image mask determine which points of the clipping area are changed, acting as an "inverse alpha" value. If the value of a source sample in the image mask is `S`, then the corresponding point in the current clipping area is multiplied by an alpha value of (`1-S`). For example, if `S` is `1` then the point in the clipping area becomes transparent. If `S` is `0`, the point in the clipping area is unchanged.

If the *mask* parameter is an image, then *mask* acts like an alpha mask and is blended with the current clipping area. The source samples of mask determine which points of the clipping area are changed. If the value of the source sample in mask is `S`, then the corresponding point in the current clipping area is multiplied by an alpha of `S`. For example, if `S` is `0`, then the point in the clipping area becomes transparent. If `S` is `1`, the point in the clipping area is unchanged.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGContext.h`

CGContextClipToRect

Sets the clipping path to the intersection of the current clipping path with the area defined by the specified rectangle.

```
void CGContextClipToRect (
   CGContextRef c,
   CGRect rect
);
```

Parameters

context

The graphics context for which to set the clipping path.

rect

A `CGRect` value that specifies, in the user space, the location and dimensions of the rectangle to be used in determining the new clipping path.

Discussion
This function sets the specified graphics context's clipping region to the area which intersects both the current clipping path and the specified rectangle.

After determining the new clipping path, the `CGContextClipToRect` function resets the context's current path to an empty path.

See also `CGContextClipToRects` (page 81).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
CGContext.h

CGContextClipToRects

Sets the clipping path to the intersection of the current clipping path with the region defined by an array of rectangles.

```
void CGContextClipToRects (
   CGContextRef c,
   const CGRect rects[],
   size_t count
);
```

Parameters

context
The graphics context for which to set the clipping path.

rects
An array of rectangles. The locations and dimensions of the rectangles are specified in the user space coordinate system.

count
The total number of array entries in the `rects` parameter.

Discussion
This function sets the clipping path to the intersection of the current clipping path and the region within the specified rectangles.

After determining the new clipping path, the function resets the context's current path to an empty path.

See also `CGContextClipToRect` (page 80).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
CGContext.h

CGContextClosePath

Closes and terminates an open path.

```
void CGContextClosePath (
   CGContextRef c
);
```

Parameters

context
A graphics context.

Discussion
If a path is open, this function closes and terminate the path. Quartz closes a path by drawing a straight line that connects the current point to the starting point. If the current point and the starting point are the same, you must still call this function to close the path. After Quartz terminates the path, the current point is no longer defined. If there is no open path, this function does nothing.

When you fill or clip an open path, Quartz implicitly closes the subpath for you.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextAddPath` (page 73)

Related Sample Code
CALayerEssentials
CarbonSketch
HID Explorer

Declared In
`CGContext.h`

CGContextConcatCTM

Transforms the user coordinate system in a context using a specified matrix.

```
void CGContextConcatCTM (
   CGContextRef c,
   CGAffineTransform transform
);
```

Parameters

context
A graphics context.

transform
The transformation matrix to apply to the specified context's current transformation matrix.

Discussion
When you call the function `CGContextConcatCTM`, it concatenates (that is, it combines) two matrices, by multiplying them together. The order in which matrices are concatenated is important, as the operations are not commutative. When you call `CGContextConcatCTM`, the resulting CTM in the context is: `CTMnew = transform * CTMcontext.`

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextConvertPointToDeviceSpace

Returns a point that is transformed from user space coordinates to device space coordinates.

```
CGPoint CGContextConvertPointToDeviceSpace (
   CGContextRef c,
   CGPoint point
);
```

Parameters

c

A graphics context.

point

The point, in user space coordinates, to transform.

Return Value

The coordinates of the point in device space coordinates.

Availability

Available in Mac OS X v10.4 and later.

See Also

CGContextConvertPointToUserSpace (page 83)

Declared In

CGContext.h

CGContextConvertPointToUserSpace

Returns a point that is transformed from device space coordinates to user space coordinates.

```
CGPoint CGContextConvertPointToUserSpace (
   CGContextRef c,
   CGPoint point
);
```

Parameters

c

A graphics context.

point

The point, in device space coordinates, to transform.

Return Value

The coordinates of the point in user space coordinates.

Availability

Available in Mac OS X v10.4 and later.

See Also

CGContextConvertPointToDeviceSpace (page 83)

Declared In

CGContext.h

CGContextConvertRectToDeviceSpace

Returns a rectangle that is transformed from user space coordinate to device space coordinates.

```
CGRect CGContextConvertRectToDeviceSpace (
   CGContextRef c,
   CGRect rect
);
```

Parameters

context

A graphics context.

rect

The rectangle, in user space coordinates, to transform.

Return Value

The rectangle in device space coordinates.

Discussion

In general affine transforms do not preserve rectangles. As a result, this function returns the smallest rectangle that contains the transformed corner points of the rectangle.

Availability

Available in Mac OS X v10.4 and later.

See Also

CGContextConvertRectToUserSpace (page 84)

Declared In

CGContext.h

CGContextConvertRectToUserSpace

Returns a rectangle that is transformed from device space coordinate to user space coordinates.

```
CGRect CGContextConvertRectToUserSpace (
   CGContextRef c,
   CGRect rect
);
```

Parameters

context

A graphics context.

rect

The rectangle, in device space coordinates, to transform.

Return Value

The rectangle in user space coordinates.

Discussion

In general, affine transforms do not preserve rectangles. As a result, this function returns the smallest rectangle that contains the transformed corner points of the rectangle.

Availability

Available in Mac OS X v10.4 and later.

See Also
CGContextConvertRectToDeviceSpace (page 84)

Declared In
CGContext.h

CGContextConvertSizeToDeviceSpace

Returns a size that is transformed from user space coordinates to device space coordinates.

```
CGSize CGContextConvertSizeToDeviceSpace (
   CGContextRef c,
   CGSize size
);
```

Parameters

c
A graphics context.

size
The size, in user space coordinates, to transform.

Return Value
The size in device space coordinates.

Availability
Available in Mac OS X v10.4 and later.

See Also
CGContextConvertSizeToUserSpace (page 85)

Declared In
CGContext.h

CGContextConvertSizeToUserSpace

Returns a size that is transformed from device space coordinates to user space coordinates

```
CGSize CGContextConvertSizeToUserSpace (
   CGContextRef c,
   CGSize size
);
```

Parameters

context
A graphics context.

size
The size, in device space coordinates, to transform.

Return Value
The size in user space coordinates.

Availability
Available in Mac OS X v10.4 and later.

See Also
CGContextConvertSizeToDeviceSpace (page 85)

Declared In
CGContext.h

CGContextDrawImage

Draws an image into a graphics context.

```
void CGContextDrawImage (
   CGContextRef c,
   CGRect rect,
   CGImageRef image
);
```

Parameters

context
The graphics context in which to draw the image.

rect
The location and dimensions in user space of the bounding box in which to draw the image.

image
The image to draw.

Discussion
Quartz scales the image—disproportionately, if necessary—to fit the bounds specified by the `rect` parameter.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonCocoa_PictureCursor
WhackedTV

Declared In
CGContext.h

CGContextDrawLinearGradient

Paints a gradient fill that varies along the line defined by the provided starting and ending points.

```
void CGContextDrawLinearGradient(
    CGContextRef context,
    CGGradientRef gradient,
    CGPoint startPoint,
    CGPoint endPoint,
    CGGradientDrawingOptions options
);
```

Parameters

context
A Quartz graphics context.

gradient
A CGGradient object.

startPoint
The coordinate that defines the starting point of the gradient.

endPoint
The coordinate that defines the ending point of the gradient.

options
Option flags (kCGGradientDrawsBeforeStartLocation (page 207) or kCGGradientDrawsAfterEndLocation (page 207)) that control whether the fill is extended beyond the starting or ending point.

Discussion
The color at location 0 in the CGGradient object is mapped to the starting point. The color at location 1 in the CGGradient object is mapped to the ending point. Colors are linearly interpolated between these two points based on the location values of the gradient. The option flags control whether the gradient is drawn before the start point or after the end point.

Availability
Available in Mac OS X v10.5 and later.

Declared In
CGContext.h

CGContextDrawPath

Draws the current path using the provided drawing mode.

```
void CGContextDrawPath (
   CGContextRef c,
   CGPathDrawingMode mode
);
```

Parameters

context
A graphics context that contains a path to paint.

mode
A path drawing mode constant—kCGPathFill, kCGPathEOFill, kCGPathStroke, kCGPathFillStroke, or kCGPathEOFillStroke. For a discussion of these constants, see *CGPath Reference*.

Discussion
This function draws the current path using the specified drawing mode. If the current path contains several disjoint portions (or subpaths), Quartz fills each one independently. Any subpath that you did not explicitly close by calling CGContextClosePath (page 81) is closed implicitly by the fill routines.

Availability
Available in Mac OS X version 10.0 and later.

See Also
CGContextFillPath (page 94)
CGContextEOFillPath (page 93)
CGContextStrokePath (page 134)

Related Sample Code
CarbonSketch

Declared In
CGContext.h

CGContextDrawPDFDocument

Draws a page of a PDF document into a graphics context.

```
void CGContextDrawPDFDocument (
   CGContextRef c,
   CGRect rect,
   CGPDFDocumentRef document,
   int page
);
```

Parameters

context
The graphics context in which to draw the PDF page.

rect
A `CGRect` value that specifies the dimensions and location of the area in which to draw the PDF page, in units of the user space. When drawn, Quartz scales the media box of the page to fit the rectangle you specify.

document
The PDF document to draw.

page
A value that specifies the PDF page number to draw. If the specified page does not exist, the function does nothing.

Special Considerations

For applications running in Mac OS X version 10.3 and later, it is recommended that you replace this function with `CGContextDrawPDFPage` (page 88). If you do so, and want to specify the drawing rectangle, you should use `CGPDFPageGetDrawingTransform` (page 355) to get an appropriate transform, concatenate it with the current transformation matrix, clip to the rectangle, and then call `CGContextDrawPDFPage` (page 88).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
CGContext.h

CGContextDrawPDFPage

Draws a page in the current user space of a PDF context.

```
void CGContextDrawPDFPage (
   CGContextRef c,
   CGPDFPageRef page
);
```

Parameters

context

The graphics context in which to draw the PDF page.

page

A Quartz PDF page.

Discussion

This function works in conjunction with the opaque type `CGPDFPageRef` to draw individual pages into a PDF context.

For applications running in Mac OS X version 10.3 and later, this function is recommended as a replacement for the older function `CGContextDrawPDFDocument`.

Availability

Available in Mac OS X version 10.3 and later.

Related Sample Code

CarbonSketch

Declared In

`CGContext.h`

CGContextDrawRadialGradient

Paints a gradient fill that varies along the area defined by the provided starting and ending circles.

```
void CGContextDrawRadialGradient(
    CGContextRef context,
    CGGradientRef gradient,
    CGPoint startCenter,
    CGFloat startRadius,
    CGPoint endCenter,
    CGFloat endRadius,
    CGGradientDrawingOptions options
);
```

Parameters

context

A Quartz graphics context.

gradient

A CGGradient object.

startCenter

The coordinate that defines the center of the starting circle.

startRadius

The radius of the starting circle.

endCenter

The coordinate that defines the center of the ending circle.

endRadius
The radius of the ending circle.

options
Option flags (`kCGGradientDrawsBeforeStartLocation` (page 207) or `kCGGradientDrawsAfterEndLocation` (page 207)) that control whether the gradient is drawn before the starting circle or after the ending circle.

Discussion
The color at location 0 in the CGGradient object is mapped to the circle defined by `startCenter` and `startRadius`. The color at location 1 in the CGGradient object is mapped to the circle defined by `endCenter` and `endRadius`. Colors are linearly interpolated between the starting and ending circles based on the location values of the gradient. The option flags control whether the gradient is drawn before the start point or after the end point.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGContext.h`

CGContextDrawShading

Fills the clipping path of a context with the specified shading.

```
void CGContextDrawShading (
   CGContextRef c,
   CGShadingRef shading
);
```

Parameters

context
The graphics context in which to draw the shading.

shading
A Quartz shading. Quartz retains this object; upon return, you may safely release it.

Discussion
In Mac OS X v10.5 and later, the preferred way to draw gradients is to use a CGGradient object. See *CGGradient Reference*.

Availability
Available in Mac OS X v10.2 and later.

See Also
`CGContextDrawLinearGradient` (page 86)
`CGContextDrawRadialGradient` (page 89)

Declared In
`CGContext.h`

CGContextDrawTiledImage

Repeatedly draws an image, scaled to the provided rectangle, to fill the current clip region.

```
void CGContextDrawTiledImage(
    CGContextRef context,
    CGRect rect,
    CGImageRef image
);
```

Parameters

context

The graphics context in which to draw the image.

rect

A rectangle that specifies the tile size. Quartz scales the image—disproportionately, if necessary—to fit the bounds specified by the `rect` parameter.

image

The image to draw.

Discussion

Quartz draws the scaled image starting at the origin of user space, then moves to a new point (horizontally by the width of the tile and/or vertically by the height of the tile), draws the scaled image, moves again, draws again, and so on, until the current clip region is tiled with copies of the image. Unlike patterns, the image is tiled in user space, so transformations applied to the CTM affect the final result.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CGContext.h`

CGContextEndPage

Ends the current page in a page-based graphics context.

```
void CGContextEndPage (
   CGContextRef c
);
```

Parameters

context

A page-based graphics context.

Discussion

When using a graphics context that supports multiple pages, you should call this function to terminate drawing in the current page.

For more information, see `CGContextBeginPage` (page 76).

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CGContext.h`

CGContextEndTransparencyLayer

Ends a transparency layer.

```
void CGContextEndTransparencyLayer (
   CGContextRef context
);
```

Parameters

context

A graphics context.

Discussion

See the discussion for `CGContextBeginTransparencyLayer` (page 77).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGContext.h`

CGContextEOClip

Modifies the current clipping path, using the even-odd rule.

```
void CGContextEOClip (
   CGContextRef c
);
```

Parameters

context

A graphics context containing a path. If the context does not have a current path, the function does nothing.

Discussion

The function uses the even-odd rule to calculate the intersection of the current path with the current clipping path. Quartz then uses the path resulting from the intersection as the new current clipping path for subsequent painting operations.

Unlike the current path, the current clipping path is part of the graphics state. Therefore, to re-enlarge the paintable area by restoring the clipping path to a prior state, you must save the graphics state before you clip and restore the graphics state after you've completed any clipped drawing.

After determining the new clipping path, the function resets the context's current path to an empty path.

See also `CGContextClip` (page 79).

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CGContext.h`

CGContextEOFillPath

Paints the area within the current path, using the even-odd fill rule.

```
void CGContextEOFillPath (
   CGContextRef c
);
```

Parameters

context

A graphics context that contains a path to fill.

Discussion

If the current path contains several disjoint portions (or subpaths), Quartz fills each one independently. Any subpath that you did not explicitly close by calling `CGContextClosePath` (page 81) is closed implicitly by the fill routines.

Availability

Available in Mac OS X version 10.0 and later.

See Also

`CGContextFillPath` (page 94)

`CGContextStrokePath` (page 134)

`CGContextDrawPath` (page 87)

Related Sample Code

CALayerEssentials

Declared In

`CGContext.h`

CGContextFillEllipseInRect

Paints the area of the ellipse that fits inside the provided rectangle, using the fill color in the current graphics state.

```
void CGContextFillEllipseInRect (
   CGContextRef context,
   CGRect rect
);
```

Parameters

context

A graphics context.

rect

A rectangle that defines the area for the ellipse to fit in.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

HID Calibrator

HID Config Save

HID Explorer

Declared In
CGContext.h

CGContextFillPath

Paints the area within the current path, using the nonzero winding number rule.

```
void CGContextFillPath (
   CGContextRef c
);
```

Parameters

context
A graphics context that contains a path to fill.

Discussion
If the current path contains several disjoint portions (or subpaths), Quartz fills each one independently. Any subpath that you did not explicitly close by calling CGContextClosePath (page 81) is closed implicitly by the fill routines.

Availability
Available in Mac OS X version 10.0 and later.

See Also
CGContextEOFillPath (page 93)
CGContextStrokePath (page 134)
CGContextDrawPath (page 87)

Declared In
CGContext.h

CGContextFillRect

Paints the area contained within the provided rectangle, using the fill color in the current graphics state.

```
void CGContextFillRect (
   CGContextRef c,
   CGRect rect
);
```

Parameters

context
A graphics context.

rect
A rectangle, in user space coordinates.

Discussion
As a side effect when you call this function, Quartz clears the current path.

Availability
Available in Mac OS X version 10.0 and later.

See Also
CGContextFillRects (page 95)

Related Sample Code
CALayerEssentials
CarbonSketch
HID Calibrator
HID Explorer

Declared In
CGContext.h

CGContextFillRects

Paints the areas contained within the provided rectangles, using the fill color in the current graphics state.

```
void CGContextFillRects (
   CGContextRef c,
   const CGRect rects[],
   size_t count
);
```

Parameters

context
A graphics context .

rects
An array of rectangles, in user space coordinates.

count
The number rectangles in the `rects` array.

Discussion
As a side effect when you call this function, Quartz clears the current path.

Availability
Available in Mac OS X version 10.0 and later.

See Also
CGContextFillRect (page 94)

Declared In
CGContext.h

CGContextFlush

Forces all pending drawing operations in a window context to be rendered immediately to the destination device.

```
void CGContextFlush (
   CGContextRef c
);
```

Parameters

context

The window context to flush. If you pass a PDF context or a bitmap context, this function does nothing.

Discussion

When you call this function, Quartz immediately flushes the current drawing to the destination device (for example, a screen). Because the system software flushes a context automatically at the appropriate times, calling this function could have an adverse effect on performance. Under normal conditions, you do not need to call this function.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextGetClipBoundingBox

Returns the bounding box of a clipping path.

```
CGRect CGContextGetClipBoundingBox (
   CGContextRef c
);
```

Parameters

context

The graphics context to modify.

Return Value

The bounding box of the clipping path, specified in user space.

Discussion

The bounding box is the smallest rectangle completely enclosing all points in the clipping path, including control points for any Bezier curves in the path.

Availability

Available in Mac OS X version 10.3 and later.

Related Sample Code

CALayerEssentials

Declared In

`CGContext.h`

CGContextGetCTM

Returns the current transformation matrix.

```
CGAffineTransform CGContextGetCTM (
   CGContextRef c
);
```

Parameters

context

A graphics context.

Return Value
The transformation matrix for the current graphics state of the specified context.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
CGContext.h

CGContextGetInterpolationQuality

Returns the current level of interpolation quality for a graphics context.

```
CGInterpolationQuality CGContextGetInterpolationQuality (
   CGContextRef c
);
```

Parameters

context

The graphics context to examine.

Return Value
The current level of interpolation quality.

Discussion
Interpolation quality is a graphics state parameter that provides a hint for the level of quality to use for image interpolation (for example, when scaling the image). Not all contexts support all interpolation quality levels.

Availability
Available in Mac OS X version 10.1 and later.

See Also
CGContextSetInterpolationQuality (page 116)

Declared In
CGContext.h

CGContextGetPathBoundingBox

Returns the smallest rectangle that contains the current path.

```
CGRect CGContextGetPathBoundingBox (
   CGContextRef c
);
```

Parameters

context

The graphics context, containing a path, to examine.

Return Value

A `CGRect` value that specifies the dimensions and location, in user space, of the bounding box of the path. If there is no path, the function returns `CGRectNull`.

Discussion

The bounding box is the smallest rectangle completely enclosing all points in a path, including control points for Bézier cubic and quadratic curves.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextGetPathCurrentPoint

Returns the current point in a non-empty path.

```
CGPoint CGContextGetPathCurrentPoint (
   CGContextRef c
);
```

Parameters

context

The graphics context containing the path to examine.

Return Value

A `CGPoint` value that specifies the location, in user space, of current point in the context's path. If there is no path, the function returns `CGPointZero`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextGetTextMatrix

Returns the current text matrix.

```
CGAffineTransform CGContextGetTextMatrix (
   CGContextRef c
);
```

Parameters

context

The graphics context for which to obtain the text matrix.

Return Value

The current text matrix.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

CGContext.h

CGContextGetTextPosition

Returns the location at which text is drawn.

```
CGPoint CGContextGetTextPosition (
   CGContextRef c
);
```

Parameters

context

The graphics context from which to obtain the current text position.

Return Value

Returns a `CGPoint` value that specifies the x and y values at which text is to be drawn, in user space coordinates.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

CGContext.h

CGContextGetTypeID

Returns the type identifier for Quartz graphics contexts.

```
CFTypeID CGContextGetTypeID (
   void
);
```

Return Value

The identifier for the opaque type `CGContextRef` (page 137).

Availability

Available in Mac OS X version 10.2 and later.

Declared In

CGContext.h

CGContextGetUserSpaceToDeviceSpaceTransform

Returns an affine transform that maps user space coordinates to device space coordinates.

```
CGAffineTransform CGContextGetUserSpaceToDeviceSpaceTransform (
   CGContextRef c
);
```

Parameters

c

A graphics context.

Return Value

The affine transforms that maps the user space of the graphics context to the device space.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGContext.h`

CGContextIsPathEmpty

Indicates whether the current path contains any subpaths.

```
bool CGContextIsPathEmpty (
   CGContextRef c
);
```

Parameters

context

The graphics context containing the path to examine.

Return Value

Returns `1` if the context's path contains no subpaths, otherwise returns `0`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextMoveToPoint

Begins a new path at the point you specify.

```
void CGContextMoveToPoint (
   CGContextRef c,
   CGFloat x,
   CGFloat y
);
```

Parameters

context

A graphics context.

x
The x-value, in user space coordinates, for the point.

y
The y-value, in user space coordinates, for the point.

Discussion
This point you specifies becomes the current point. It defines the starting point of the next line segment.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CALayerEssentials
CarbonSketch
HID Calibrator
HID Explorer

Declared In
`CGContext.h`

CGContextPathContainsPoint

Checks to see whether the specified point is contained in the current path.

```
bool CGContextPathContainsPoint (
   CGContextRef context,
   CGPoint point,
   CGPathDrawingMode mode
);
```

Parameters

context
A graphics context.

point
The point to check, specified in user space units.

mode
A path drawing mode—`kCGPathFill`, `kCGPathEOFill`, `kCGPathStroke`, `kCGPathFillStroke`, or `kCGPathEOFillStroke`. See `CGPathDrawingMode` for more information on these modes.

Return Value
Returns `true` if `point` is inside the current path of the graphics context; `false` otherwise.

Discussion
A point is contained within the path of a graphics context if the point is inside the painted region when the path is stroked or filled with opaque colors using the specified path drawing mode. A point can be inside a path only if the path is explicitly closed by calling the function `CGContextClosePath` (page 81), for paths drawn directly to the current context, or `CGPathCloseSubpath` (page 271), for paths first created as `CGPath` objects and then drawn to the current context.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGContext.h

CGContextRelease

Decrements the retain count of a graphics context.

```
void CGContextRelease (
   CGContextRef c
);
```

Parameters

context
The graphics context to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `context` parameter is `NULL`.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
CGContext.h

CGContextReplacePathWithStrokedPath

Replaces the path in the graphics context with the stroked version of the path.

```
void CGContextReplacePathWithStrokedPath (
   CGContextRef c
);
```

Parameters

c
A graphics context.

Discussion
Quartz creates a stroked path using the parameters of the current graphics context. You can use this path in the same way you use the path of any context. For example, you can clip to the stroked version of a path by calling this function followed by a call to the function `CGContextClip` (page 79).

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGContext.h

CGContextRestoreGState

Sets the current graphics state to the state most recently saved.

```
void CGContextRestoreGState (
   CGContextRef c
);
```

Parameters

context

The graphics context whose state you want to modify.

Discussion

Quartz removes the graphics state that is at the top of the stack so that the most recently saved state becomes the current graphics state.

Availability

Available in Mac OS X version 10.0 and later.

See Also

`CGContextSaveGState` (page 104)

Related Sample Code

CarbonSketch

HID Calibrator

Declared In

`CGContext.h`

CGContextRetain

Increments the retain count of a graphics context.

```
CGContextRef CGContextRetain (
   CGContextRef c
);
```

Parameters

context

The graphics context to retain.

Return Value

The same graphics context you passed in as the `context` parameter.

Discussion

This function is equivalent to `CFRetain`, except that it does not cause an error if the `context` parameter is `NULL`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextRotateCTM

Rotates the user coordinate system in a context.

```
void CGContextRotateCTM (
   CGContextRef c,
   CGFloat angle
);
```

Parameters

context

A graphics context.

angle

The angle, in radians, by which to rotate the coordinate space of the specified context. (Positive values rotate counterclockwise.)

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSaveGState

Pushes a copy of the current graphics state onto the graphics state stack for the context.

```
void CGContextSaveGState (
   CGContextRef c
);
```

Parameters

context

The graphics context whose current graphics state you want to save.

Discussion

Each graphics context maintains a stack of graphics states. Note that not all aspects of the current drawing environment are elements of the graphics state. For example, the current path is not considered part of the graphics state and is therefore not saved when you call the `CGContextSaveGState` function. The graphics state parameters that *are* saved are:

- CTM (current transformation matrix)
- clip region
- image interpolation quality
- line width
- line join
- miter limit
- line cap
- line dash
- flatness
- should anti-alias

- rendering intent
- fill color space
- stroke color space
- fill color
- stroke color
- alpha value
- font
- font size
- character spacing
- text drawing mode
- shadow parameters
- the pattern phase
- the font smoothing parameter
- blend mode

To restore your drawing environment to a previously saved state, you can use the function `CGContextRestoreGState` (page 103).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch
HID Calibrator

Declared In
`CGContext.h`

CGContextScaleCTM

Changes the scale of the user coordinate system in a context.

```
void CGContextScaleCTM (
   CGContextRef c,
   CGFloat sx,
   CGFloat sy
);
```

Parameters

context
A graphics context.

sx
The factor by which to scale the x-axis of the coordinate space of the specified context.

sy
The factor by which to scale the y-axis of the coordinate space of the specified context.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
CGContext.h

CGContextSelectFont

Sets the font and font size in a graphics context.

```
void CGContextSelectFont (
   CGContextRef c,
   const char *name,
   CGFloat size,
   CGTextEncoding textEncoding
);
```

Parameters

context
The graphics context for which to set the font and font size.

name
A null-terminated string that contains the PostScript name of the font to set.

size
A value that specifies the font size to set, in text space units.

textEncoding
A `CGTextEncoding` value that specifies the encoding used for the font. For a description of the available values, see "Text Encodings" (page 146).

Discussion
For information about when to use this function, see `CGContextShowText` (page 131) and `CGContextShowTextAtPoint` (page 132).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
HID Calibrator

Declared In
CGContext.h

CGContextSetAllowsAntialiasing

Sets whether or not to allow anti-aliasing for a graphics context.

```
void CGContextSetAllowsAntialiasing (
   CGContextRef context,
   bool allowsAntialiasing
);
```

Parameters

context

A graphics context.

allowsAntialiasing

A Boolean value that specifies whether or not to allow antialiasing. Pass `true` to allow antialiasing; `false` otherwise. This parameter is not part of the graphics state.

Discussion

Quartz performs antialiasing for a graphics context if both the `allowsAntialiasing` parameter and the graphics state parameter `shouldAntialias` are `true`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGContext.h`

CGContextSetAlpha

Sets the opacity level for objects drawn in a graphics context.

```
void CGContextSetAlpha (
   CGContextRef c,
   CGFloat alpha
);
```

Parameters

context

The graphics context for which to set the current graphics state's alpha value parameter.

alpha

A value that specifies the opacity level. Values can range from `0.0` (transparent) to `1.0` (opaque). Values outside this range are clipped to `0.0` or `1.0`.

Discussion

This function sets the alpha value parameter for the specified graphics context. To clear the contents of the drawing canvas, you should use the function `CGContextClearRect` (page 78).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSetBlendMode

Sets how Quartz composites sample values for a graphics context.

```
void CGContextSetBlendMode (
   CGContextRef context,
   CGBlendMode mode
);
```

Parameters

context

The graphics context to modify.

mode

A blend mode. See "Blend Modes" (page 137) for a list of the constants you can supply.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGContext.h

CGContextSetCharacterSpacing

Sets the current character spacing.

```
void CGContextSetCharacterSpacing (
   CGContextRef c,
   CGFloat spacing
);
```

Parameters

context

The graphics context for which to set the character spacing.

spacing

A value that represents the amount of additional space to place between glyphs, in text space coordinates.

Discussion
Quartz adds the additional space to the advance between the origin of one character and the origin of the next character. For information about the text coordinate system, see `CGContextSetTextMatrix` (page 127).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
CGContext.h

CGContextSetCMYKFillColor

Sets the current fill color to a value in the DeviceCMYK color space.

```
void CGContextSetCMYKFillColor (
   CGContextRef c,
   CGFloat cyan,
   CGFloat magenta,
   CGFloat yellow,
   CGFloat black,
   CGFloat alpha
);
```

Parameters

context

The graphics context for which to set the current fill color.

cyan

The cyan intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from 0.0 (does not absorb the secondary color) to 1.0 (fully absorbs the secondary color).

magenta

The magenta intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from 0.0 (does not absorb the secondary color) to 1.0 (fully absorbs the secondary color).

yellow

The yellow intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from 0.0 (does not absorb the secondary color) to 1.0 (fully absorbs the secondary color).

black

The black intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from 0.0 (does not absorb the secondary color) to 1.0 (fully absorbs the secondary color).

alpha

A value that specifies the opacity level. Values can range from `0.0` (transparent) to `1.0` (opaque). Values outside this range are clipped to `0.0` or `1.0`.

Discussion

Quartz provides convenience functions for each of the device color spaces that allow you to set the fill or stroke color space and the fill or stroke color with one function call.

When you call this function, two things happen:

- Quartz sets the current fill color space to DeviceCMYK.
- Quartz sets the current fill color to the value specified by the `cyan`, `magenta`, `yellow`, `black`, and `alpha` parameters.

See also `CGContextSetCMYKStrokeColor` (page 110).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSetCMYKStrokeColor

Sets the current stroke color to a value in the DeviceCMYK color space.

```
void CGContextSetCMYKStrokeColor (
   CGContextRef c,
   CGFloat cyan,
   CGFloat magenta,
   CGFloat yellow,
   CGFloat black,
   CGFloat alpha
);
```

Parameters

context

The graphics context for which to set the current stroke color.

cyan

The cyan intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from `0.0` (does not absorb the secondary color) to `1.0` (fully absorbs the secondary color).

magenta

The magenta intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from `0.0` (does not absorb the secondary color) to `1.0` (fully absorbs the secondary color).

yellow

The yellow intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from `0.0` (does not absorb the secondary color) to `1.0` (fully absorbs the secondary color).

black

The black intensity value for the color to set. The DeviceCMYK color space permits the specification of a value ranging from `0.0` (does not absorb the secondary color) to `1.0` (fully absorbs the secondary color).

alpha

A value that specifies the opacity level. Values can range from `0.0` (transparent) to `1.0` (opaque). Values outside this range are clipped to `0.0` or `1.0`.

Discussion

When you call this function, two things happen:

- Quartz sets the current stroke color space to DeviceCMYK.
- Quartz sets the current stroke color to the value specified by the `cyan`, `magenta`, `yellow`, `black`, and `alpha` parameters.

See also `CGContextSetCMYKFillColor` (page 108).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSetFillColor

Sets the current fill color.

```
void CGContextSetFillColor (
   CGContextRef c,
   const CGFloat components[]
);
```

Parameters

context

The graphics context for which to set the current fill color.

components

An array of intensity values describing the color to set. The number of array elements must equal the number of components in the current fill color space, plus an additional component for the alpha value.

Discussion

The current fill color space must not be a pattern color space. For information on setting the fill color when using a pattern color space, see `CGContextSetFillPattern` (page 112). Note that the preferred API to use is now `CGContextSetFillColorWithColor` (page 112).

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CGContext.h`

CGContextSetFillColorSpace

Sets the fill color space in a graphics context.

```
void CGContextSetFillColorSpace (
   CGContextRef c,
   CGColorSpaceRef colorspace
);
```

Parameters

context

The graphics context for which to set the fill color space.

colorspace

The new fill color space. Quartz retains this object; upon return, you may safely release it.

Discussion

As a side effect of this function, Quartz assigns an appropriate initial value to the fill color, based on the specified color space. To change this value, call `CGContextSetFillColor` (page 111). Note that the preferred API to use is now `CGContextSetFillColorWithColor` (page 112).

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextSetFillColorWithColor

Sets the current fill color in a graphics context, using a Quartz color.

```
void CGContextSetFillColorWithColor (
   CGContextRef c,
   CGColorRef color
);
```

Parameters

context
The graphics context for which to set the fill color.

color
The new fill color.

Discussion
See also `CGContextSetFillColor` (page 111).

Availability
Available in Mac OS X version 10.3 and later.

Related Sample Code
CALayerEssentials

Declared In
`CGContext.h`

CGContextSetFillPattern

Sets the fill pattern in the specified graphics context.

```
void CGContextSetFillPattern (
   CGContextRef c,
   CGPatternRef pattern,
   const CGFloat components[]
);
```

Parameters

context
The graphics context to modify.

pattern
A fill pattern. Quartz retains this object; upon return, you may safely release it.

components

If the pattern is an uncolored (or a masking) pattern, pass an array of intensity values that specify the color to use when the pattern is painted. The number of array elements must equal the number of components in the base space of the fill pattern color space, plus an additional component for the alpha value.

If the pattern is a colored pattern, pass an alpha value.

Discussion

The current fill color space must be a pattern color space. Otherwise, the result of calling this function is undefined. If you want to set a fill color, not a pattern, then call the function `CGContextSetFillColorWithColor` (page 112).

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CGContext.h`

CGContextSetFlatness

Sets the accuracy of curved paths in a graphics context.

```
void CGContextSetFlatness (
   CGContextRef c,
   CGFloat flatness
);
```

Parameters

context

The graphics context to modify.

flatness

The largest permissible distance, measured in device pixels, between a point on the true curve and a point on the approximated curve.

Discussion

This function controls how accurately curved paths are rendered. Setting the flatness value to less than `1.0` renders highly accurate curves, but lengthens rendering times.

In most cases, you should not change the flatness value. Customizing the flatness value for the capabilities of a particular output device impairs the ability of your application to render to other devices.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSetFont

Sets the platform font in a graphics context.

```
void CGContextSetFont (
   CGContextRef c,
   CGFontRef font
);
```

Parameters

context

The graphics context for which to set the font.

font

A Quartz font.

Discussion

For information about when to use this function, see `CGFontCreateWithPlatformFont` (page 183).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSetFontSize

Sets the current font size.

```
void CGContextSetFontSize (
   CGContextRef c,
   CGFloat size
);
```

Parameters

context

A graphics context.

size

A font size, expressed in text space units.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSetGrayFillColor

Sets the current fill color to a value in the DeviceGray color space.

```
void CGContextSetGrayFillColor (
   CGContextRef c,
   CGFloat gray,
   CGFloat alpha
);
```

Parameters

context

The graphics context for which to set the current fill color.

gray

A value that specifies the desired gray level. The DeviceGray color space permits the specification of a value ranging from `0.0` (absolute black) to `1.0` (absolute white). Values outside this range are clamped to `0.0` or `1.0`.

alpha

A value that specifies the opacity level. Values can range from `0.0` (transparent) to `1.0` (opaque). Values outside this range are clipped to `0.0` or `1.0`.

Discussion
When you call this function, two things happen:

- Quartz sets the current fill color space to DeviceGray.
- Quartz sets the current fill color to the value you specify in the `gray` and `alpha` parameters.

See also `CGContextSetGrayStrokeColor` (page 115).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGContext.h`

CGContextSetGrayStrokeColor

Sets the current stroke color to a value in the DeviceGray color space.

```
void CGContextSetGrayStrokeColor (
   CGContextRef c,
   CGFloat gray,
   CGFloat alpha
);
```

Parameters

context

The graphics context for which to set the current stroke color.

gray

A value that specifies the desired gray level. The DeviceGray color space permits the specification of a value ranging from `0.0` (absolute black) to `1.0` (absolute white). Values outside this range are clamped to `0.0` or `1.0`.

alpha

A value that specifies the opacity level. Values can range from `0.0` (transparent) to `1.0` (opaque). Values outside this range are clipped to `0.0` or `1.0`.

Discussion
When you call this function, two things happen:

- Quartz sets the current stroke color space to DeviceGray. The DeviceGray color space is a single-dimension space in which color values are specified solely by the intensity of a gray value (from absolute black to absolute white).
- Quartz sets the current stroke color to the value you specify in the `gray` and `alpha` parameters.

See also `CGContextSetGrayFillColor` (page 114).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGContext.h`

CGContextSetInterpolationQuality

Sets the level of interpolation quality for a graphics context.

```
void CGContextSetInterpolationQuality (
   CGContextRef c,
   CGInterpolationQuality quality
);
```

Parameters

context
The graphics context to modify.

quality
A `CGInterpolationQuality` constant that specifies the required level of interpolation quality. For possible values, see "`Interpolation Qualities`" (page 142).

Discussion
Interpolation quality is merely a hint to the context—not all contexts support all interpolation quality levels.

Availability
Available in Mac OS X version 10.1 and later.

See Also
`CGContextGetInterpolationQuality` (page 97)

Declared In
`CGContext.h`

CGContextSetLineCap

Sets the style for the endpoints of lines drawn in a graphics context.

```
void CGContextSetLineCap (
   CGContextRef c,
   CGLineCap cap
);
```

Parameters

context

The graphics context to modify.

cap

A line cap style constant—`kCGLineCapButt` (page 143) (the default), `kCGLineCapRound` (page 143), or `kCGLineCapSquare` (page 143). See "`Line Cap Styles`" (page 143).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextSetLineDash

Sets the pattern for dashed lines in a graphics context.

```
void CGContextSetLineDash (
   CGContextRef c,
   CGFloat phase,
   const CGFloat lengths[],
   size_t count
);
```

Parameters

context

The graphics context to modify.

phase

A value that specifies how far into the dash pattern the line starts, in units of the user space. For example, passing a value of `3` means the line is drawn with the dash pattern starting at three units from its beginning. Passing a value of `0` draws a line starting with the beginning of a dash pattern.

lengths

An array of values that specify the lengths of the painted segments and unpainted segments, respectively, of the dash pattern—or `NULL` for no dash pattern.

For example, passing an array with the values `[2,3]` sets a dash pattern that alternates between a 2-user-space-unit-long painted segment and a 3-user-space-unit-long unpainted segment. Passing the values `[1,3,4,2]` sets the pattern to a 1-unit painted segment, a 3-unit unpainted segment, a 4-unit painted segment, and a 2-unit unpainted segment.

count

If the `lengths` parameter specifies an array, pass the number of elements in the array. Otherwise, pass `0`.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextSetLineJoin

Sets the style for the joins of connected lines in a graphics context.

```
void CGContextSetLineJoin (
   CGContextRef c,
   CGLineJoin join
);
```

Parameters

context
The graphics context to modify.

join
A line join value—`kCGLineJoinMiter` (page 144) (the default), `kCGLineJoinRound` (page 144), or `kCGLineJoinBevel` (page 144). See `"Line Joins"` (page 144).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextSetLineWidth

Sets the line width for a graphics context.

```
void CGContextSetLineWidth (
   CGContextRef c,
   CGFloat width
);
```

Parameters

context
The graphics context to modify.

width
The new line width to use, in user space units. The value must be greater than `0`.

Discussion
The default line width is `1` unit. When stroked, the line straddles the path, with half of the total width on either side.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextSetMiterLimit

Sets the miter limit for the joins of connected lines in a graphics context.

```
void CGContextSetMiterLimit (
   CGContextRef c,
   CGFloat limit
);
```

Parameters

context
The graphics context to modify.

limit
The miter limit to use.

Discussion
If the current line join style is set to `kCGLineJoinMiter` (see `CGContextSetLineJoin` (page 118)), Quartz uses the miter limit to determine whether the lines should be joined with a bevel instead of a miter. Quartz divides the length of the miter by the line width. If the result is greater than the miter limit, Quartz converts the style to a bevel.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGContext.h`

CGContextSetPatternPhase

Sets the pattern phase of a context.

```
void CGContextSetPatternPhase (
   CGContextRef c,
   CGSize phase
);
```

Parameters

context
The graphics context to modify.

phase
A pattern phase, specified in user space.

Discussion
The pattern phase is a translation that Quartz applies prior to drawing a pattern in the context. The pattern phase is part of the graphics state of a context, and the default pattern phase is `(0,0)`. Setting the pattern phase has the effect of temporarily changing the pattern matrix of any pattern you draw. For example, setting the context's pattern phase to `(2,3)` has the effect of moving the start of pattern cell tiling to the point `(2,3)` in default user space.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGContext.h`

CGContextSetRenderingIntent

Sets the rendering intent in the current graphics state.

```
void CGContextSetRenderingIntent (
   CGContextRef c,
   CGColorRenderingIntent intent
);
```

Parameters

context
The graphics context to modify.

intent
A rendering intent constant—`kCGRenderingIntentDefault` (page 59), `kCGRenderingIntentAbsoluteColorimetric` (page 59), `kCGRenderingIntentRelativeColorimetric` (page 59), `kCGRenderingIntentPerceptual` (page 59), or `kCGRenderingIntentSaturation` (page 59). For a discussion of these constants, see *CGColorSpace Reference*.

Discussion
The rendering intent specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context. If you do not explicitly set the rendering intent, Quartz uses perceptual rendering intent for drawing sampled images and relative colorimetric rendering intent for all other drawing.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGContext.h`

CGContextSetRGBFillColor

Sets the current fill color to a value in the DeviceRGB color space.

```
void CGContextSetRGBFillColor (
   CGContextRef c,
   CGFloat red,
   CGFloat green,
   CGFloat blue,
   CGFloat alpha
);
```

Parameters

context

The graphics context for which to set the current fill color.

red

The red intensity value for the color to set. The DeviceRGB color space permits the specification of a value ranging from `0.0` (zero intensity) to `1.0` (full intensity).

green

The green intensity value for the color to set. The DeviceRGB color space permits the specification of a value ranging from `0.0` (zero intensity) to `1.0` (full intensity).

blue

The blue intensity value for the color to set. The DeviceRGB color space permits the specification of a value ranging from `0.0` (zero intensity) to `1.0` (full intensity).

alpha

A value that specifies the opacity level. Values can range from `0.0` (transparent) to `1.0` (opaque). Values outside this range are clipped to `0.0` or `1.0`.

Discussion

When you call this function, two things happen:

- Quartz sets the current fill color space to DeviceRGB.
- Quartz sets the current fill color to the value specified by the `red`, `green`, `blue`, and `alpha` parameters.

See also `CGContextSetRGBStrokeColor` (page 121).

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CALayerEssentials
CarbonSketch
HID Calibrator
HID Config Save
HID Explorer

Declared In

`CGContext.h`

CGContextSetRGBStrokeColor

Sets the current stroke color to a value in the DeviceRGB color space.

```
void CGContextSetRGBStrokeColor (
   CGContextRef c,
   CGFloat red,
   CGFloat green,
   CGFloat blue,
   CGFloat alpha
);
```

Parameters

context

The graphics context for which to set the current stroke color.

red

The red intensity value for the color to set. The DeviceRGB color space permits the specification of a value ranging from `0.0` (zero intensity) to `1.0` (full intensity).

green

The green intensity value for the color to set. The DeviceRGB color space permits the specification of a value ranging from `0.0` (zero intensity) to `1.0` (full intensity).

blue

The blue intensity value for the color to set. The DeviceRGB color space permits the specification of a value ranging from `0.0` (zero intensity) to `1.0` (full intensity).

alpha

A value that specifies the opacity level. Values can range from `0.0` (transparent) to `1.0` (opaque). Values outside this range are clipped to `0.0` or `1.0`.

Discussion

When you call this function, two things happen:

- Quartz sets the current stroke color space to DeviceRGB.
- Quartz sets the current stroke color to the value specified by the `red`, `green`, `blue`, and `alpha` parameters.

See also `CGContextSetRGBFillColor` (page 120).

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CarbonSketch
HID Calibrator
HID Config Save
HID Explorer

Declared In

`CGContext.h`

CGContextSetShadow

Enables shadowing in a graphics context.

```
void CGContextSetShadow (
   CGContextRef context,
   CGSize offset,
   CGFloat blur
);
```

Parameters

context

A graphics context.

offset

Specifies a translation of the context's coordinate system, to establish an offset for the shadow (`{0,0}` specifies a light source immediately above the screen).

blur

A non-negative number specifying the amount of blur.

Discussion

Shadow parameters are part of the graphics state in a context. After shadowing is set, all objects drawn are shadowed using a black color with 1/3 alpha (i.e., RGBA = `{0, 0, 0, 1.0/3.0}`) in the DeviceRGB color space.

To turn off shadowing:

- Use the standard save/restore mechanism for the graphics state.
- Use `CGContextSetShadowWithColor` (page 123) to set the shadow color to a fully transparent color (or pass `NULL` as the color).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGContext.h`

CGContextSetShadowWithColor

Enables shadowing with color a graphics context.

```
void CGContextSetShadowWithColor (
   CGContextRef context,
   CGSize offset,
   CGFloat blur,
   CGColorRef color
);
```

Parameters

context

The graphics context to modify.

offset

Specifies a translation in base-space.

blur

A non-negative number specifying the amount of blur.

color
Specifies the color of the shadow, which may contain a non-opaque alpha value. If `NULL`, then shadowing is disabled.

Discussion
See also `CGContextSetShadow` (page 122).

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGContext.h`

CGContextSetShouldAntialias

Sets anti-aliasing on or off for a graphics context.

```
void CGContextSetShouldAntialias (
   CGContextRef c,
   bool shouldAntialias
);
```

Parameters

context
The graphics context to modify.

shouldAntialias
A Boolean value that specifies whether anti-aliasing should be turned on. Anti-aliasing is turned on by default when a window or bitmap context is created. It is turned off for other types of contexts.

Discussion
Anti-aliasing is a graphics state parameter.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGContext.h`

CGContextSetShouldSmoothFonts

Enables or disables font smoothing in a graphics context.

```
void CGContextSetShouldSmoothFonts (
   CGContextRef c,
   bool shouldSmoothFonts
);
```

Parameters

context
The graphics context to modify.

shouldSmoothFonts
A Boolean value that specifies whether to enable font smoothing.

Discussion
There are cases, such as rendering to a bitmap, when font smoothing is not appropriate and should be disabled. Note that some contexts (such as PostScript contexts) do not support font smoothing.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGContext.h`

CGContextSetStrokeColor

Sets the current stroke color.

```
void CGContextSetStrokeColor (
   CGContextRef c,
   const CGFloat components[]
);
```

Parameters

context

The graphics context for which to set the current stroke color.

components

An array of intensity values describing the color to set. The number of array elements must equal the number of components in the current stroke color space, plus an additional component for the alpha value.

Discussion
The current stroke color space must not be a pattern color space. For information on setting the stroke color when using a pattern color space, see `CGContextSetStrokePattern` (page 126). Note that the preferred API is now `CGContextSetStrokeColorWithColor` (page 126).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextSetStrokeColorSpace

Sets the stroke color space in a graphics context.

```
void CGContextSetStrokeColorSpace (
   CGContextRef c,
   CGColorSpaceRef colorspace
);
```

Parameters

context

The graphics context for the new stroke color space.

colorspace
The new stroke color space. Quartz retains this object; upon return, you may safely release it.

Discussion
As a side effect when you call this function, Quartz assigns an appropriate initial value to the stroke color, based on the color space you specify. To change this value, call `CGContextSetStrokeColor` (page 125). Note that the preferred API is now `CGContextSetStrokeColorWithColor` (page 126).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

CGContextSetStrokeColorWithColor

Sets the current stroke color in a context, using a Quartz color.

```
void CGContextSetStrokeColorWithColor (
   CGContextRef c,
   CGColorRef color
);
```

Parameters

context
The graphics context to modify.

color
The new stroke color.

Discussion
See also `CGContextSetStrokeColor` (page 125).

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGContext.h`

CGContextSetStrokePattern

Sets the stroke pattern in the specified graphics context.

```
void CGContextSetStrokePattern (
   CGContextRef c,
   CGPatternRef pattern,
   const CGFloat components[]
);
```

Parameters

context
The graphics context to modify.

pattern

A pattern for stroking. Quartz retains this object; upon return, you may safely release it.

components

If the specified pattern is an uncolored (or masking) pattern, pass an array of intensity values that specify the color to use when the pattern is painted. The number of array elements must equal the number of components in the base space of the stroke pattern color space, plus an additional component for the alpha value.

If the specified pattern is a colored pattern, pass an alpha value.

Discussion

The current stroke color space must be a pattern color space. Otherwise, the result of calling this function is undefined. If you want to set a stroke color, not a stroke pattern, then call the function `CGContextSetStrokeColorWithColor` (page 126).

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CGContext.h`

CGContextSetTextDrawingMode

Sets the current text drawing mode.

```
void CGContextSetTextDrawingMode (
   CGContextRef c,
   CGTextDrawingMode mode
);
```

Parameters

context

A graphics context.

mode

A text drawing mode (such as `kCGTextFill` (page 145) or `kCGTextStroke` (page 145)) that specifies how Quartz renders individual glyphs in a graphics context. See "`Text Drawing Modes`" (page 144) for a complete list.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextSetTextMatrix

Sets the current text matrix.

```
void CGContextSetTextMatrix (
   CGContextRef c,
   CGAffineTransform t
);
```

Parameters

context

A graphics context.

transform

The text matrix to set.

Discussion

The text matrix specifies the transform from text space to user space. To produce the final text rendering matrix that is used to actually draw the text on the page, Quartz concatenates the text matrix with the current transformation matrix and other parameters from the graphics state.

Note that the text matrix is *not* a part of the graphics state—saving or restoring the graphics state has no effect on the text matrix. The text matrix is an attribute of the graphics context, not of the current font.

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

HID Calibrator

Declared In

`CGContext.h`

CGContextSetTextPosition

Sets the location at which text is drawn.

```
void CGContextSetTextPosition (
   CGContextRef c,
   CGFloat x,
   CGFloat y
);
```

Parameters

context

A graphics context.

x

A value for the x-coordinate at which to draw the text, in user space coordinates.

y

A value for the y-coordinate at which to draw the text.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextShowGlyphs

Displays an array of glyphs at the current text position.

```
void CGContextShowGlyphs (
   CGContextRef c,
   const CGGlyph g[],
   size_t count
);
```

Parameters

context

The graphics context in which to display the glyphs.

glyphs

An array of glyphs to display.

count

The total number of glyphs passed in the `g` parameter.

Discussion

This function displays an array of glyphs at the current text position, a point specified by the current text matrix.

See also `CGContextShowGlyphsAtPoint` (page 129), `CGContextShowText` (page 131), `CGContextShowTextAtPoint` (page 132), and `CGContextShowGlyphsWithAdvances` (page 130).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGContext.h`

CGContextShowGlyphsAtPoint

Displays an array of glyphs at a position you specify.

```
void CGContextShowGlyphsAtPoint (
   CGContextRef c,
   CGFloat x,
   CGFloat y,
   const CGGlyph glyphs[],
   size_t count
);
```

Parameters

context

The graphics context in which to display the glyphs.

x

A value for the x-coordinate of the user space at which to display the glyphs.

y

A value for the y-coordinate of the user space at which to display the glyphs.

glyphs

An array of glyphs to display.

count
The total number of glyphs passed in the `glyphs` parameter.

Discussion
This function displays an array of glyphs at the specified position in the text space.

See also `CGContextShowText` (page 131), `CGContextShowGlyphs` (page 129), `CGContextShowGlyphs` (page 129), and `CGContextShowGlyphsWithAdvances` (page 130).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGContext.h`

CGContextShowGlyphsAtPositions

Draws glyphs at the provided position.

```
void CGContextShowGlyphsAtPositions(
    CGContextRef context,
    const CGGlyph glyphs[],
    const CGPoint positions[],
    size_t count
);
```

Parameters
context
The graphics context in which to display the glyphs.

glyphs
An array of Quartz glyphs.

positions
The positions for the glyphs. Each item in this array matches with the glyph at the corresponding index in the `glyphs` array. The position of each glyph is specified in text space, and, as a consequence, is transformed through the text matrix to user space.

count
The number of items in the `glyphs` array.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGContext.h`

CGContextShowGlyphsWithAdvances

Draws an array of glyphs with varying offsets.

```
void CGContextShowGlyphsWithAdvances (
   CGContextRef c,
   const CGGlyph glyphs[],
   const CGSize advances[],
   size_t count
);
```

Parameters

context

The graphics context in which to display the glyphs.

glyphs

An array of Quartz glyphs.

advances

An array of offset values associated with each glyph in the array. Each value specifies the offset from the previous glyph's origin to the origin of the corresponding glyph. Offsets are specified in user space.

count

The number of glyphs in the specified array.

Discussion

This function draws an array of glyphs at the current point specified by the text matrix.

See also `CGContextShowText` (page 131), `CGContextShowGlyphs` (page 129), and `CGContextShowGlyphs` (page 129), and `CGContextShowGlyphsAtPoint` (page 129).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGContext.h`

CGContextShowText

Displays a character array at the current text position, a point specified by the current text matrix.

```
void CGContextShowText (
   CGContextRef c,
   const char *string,
   size_t length
);
```

Parameters

context

A graphics context.

string

An array of characters to draw.

length

The length of the array specified in the `bytes` parameter.

Discussion
Quartz uses font data provided by the system to map each byte of the array through the encoding vector of the current font to obtain the glyph to display. Note that the font must have been set using `CGContextSelectFont` (page 106). Don't use `CGContextShowTextAtPoint` in conjunction with `CGContextSetFont` (page 113).

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextShowTextAtPoint` (page 132)
`CGContextShowGlyphs` (page 129)
`CGContextShowGlyphsAtPoint` (page 129)
`CGContextShowGlyphsWithAdvances` (page 130)

Declared In
`CGContext.h`

CGContextShowTextAtPoint

Displays a character string at a position you specify.

```
void CGContextShowTextAtPoint (
   CGContextRef c,
   CGFloat x,
   CGFloat y,
   const char *string,
   size_t length
);
```

Parameters

context
A graphics context .

x
A value for the x-coordinate of the text space at which to display the text.

y
A value for the y-coordinate of the text space at which to display the text.

string
An array of characters to draw.

length
The length of the array specified in the `bytes` parameter.

Discussion
Quartz uses font data provided by the system to map each byte of the array through the encoding vector of the current font to obtain the glyph to display. Note that the font must have been set using `CGContextSelectFont` (page 106). Don't use `CGContextShowTextAtPoint` in conjunction with `CGContextSetFont` (page 113).

Availability
Available in Mac OS X version 10.0 and later.

See Also
CGContextShowText (page 131)
CGContextShowGlyphs (page 129)
CGContextShowGlyphsAtPoint (page 129)
CGContextShowGlyphsWithAdvances (page 130)

Related Sample Code
HID Calibrator

Declared In
CGContext.h

CGContextStrokeEllipseInRect

Strokes an ellipse that fits inside the specified rectangle.

```
void CGContextStrokeEllipseInRect (
   CGContextRef context,
   CGRect rect
);
```

Parameters

context
A graphics context.

rect
A rectangle that defines the area for the ellipse to fit in.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGContext.h

CGContextStrokeLineSegments

Strokes a sequence of line segments.

```
void CGContextStrokeLineSegments (
   CGContextRef c,
   const CGPoint points[],
   size_t count
);
```

Parameters

c
A graphics context.

points
An array of points, organized as pairs—the starting point of a line segment followed by the ending point of a line segment. For example, the first point in the array specifies the starting position of the first line, the second point specifies the ending position of the first line, the third point specifies the starting position of the second line, and so forth.

count

The number of points in the `points` array.

Discussion
This function is equivalent to the following code:

```
CGContextBeginPath (context);
for (k = 0; k < count; k += 2) {
    CGContextMoveToPoint(context, s[k].x, s[k].y);
    CGContextAddLineToPoint(context, s[k+1].x, s[k+1].y);
}
CGContextStrokePath(context);
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGContext.h`

CGContextStrokePath

Paints a line along the current path.

```
void CGContextStrokePath (
   CGContextRef c
);
```

Parameters

context

A graphics context.

Discussion
Quartz uses the line width and stroke color of the graphics state to paint the path. As a side effect when you call this function, Quartz clears the current path.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextDrawPath` (page 87)
`CGContextFillPath` (page 94)
`CGContextEOFillPath` (page 93)

Related Sample Code
CarbonSketch
HID Calibrator
HID Explorer

Declared In
`CGContext.h`

CGContextStrokeRect

Paints a rectangular path.

```
void CGContextStrokeRect (
   CGContextRef c,
   CGRect rect
);
```

Parameters

context

A graphics context .

rect

A rectangle, specified in user space coordinates.

Discussion
Quartz uses the line width and stroke color of the graphics state to paint the path.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextStrokeRectWithWidth` (page 135)

Related Sample Code
CarbonSketch
HID Calibrator
HID Config Save

Declared In
`CGContext.h`

CGContextStrokeRectWithWidth

Paints a rectangular path, using the specified line width.

```
void CGContextStrokeRectWithWidth (
   CGContextRef c,
   CGRect rect,
   CGFloat width
);
```

Parameters

context

A graphics context.

rect

A rectangle, in user space coordinates.

width

A value, in user space units, that is greater than zero. This value does not affect the line width values in the current graphics state.

Discussion
Aside from the line width value, Quartz uses the current attributes of the graphics state (such as stroke color) to paint the line. The line straddles the path, with half of the total width on either side. As a side effect when you call this function, Quartz clears the current path.

Availability
Available in Mac OS X version 10.0 and later.

See Also
CGContextStrokeRect (page 134)

Declared In
CGContext.h

CGContextSynchronize

Marks a window context for update.

```
void CGContextSynchronize (
   CGContextRef c
);
```

Parameters

context

The window context to synchronize. If you pass a PDF context or a bitmap context, this function does nothing.

Discussion
When you call this function, all drawing operations since the last update are flushed at the next regular opportunity. Under normal conditions, you do not need to call this function.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
CGContext.h

CGContextTranslateCTM

Changes the origin of the user coordinate system in a context.

```
void CGContextTranslateCTM (
   CGContextRef c,
   CGFloat tx,
   CGFloat ty
);
```

Parameters

context

A graphics context.

tx

The amount to displace the x-axis of the coordinate space, in units of the user space, of the specified context.

ty

The amount to displace the y-axis of the coordinate space, in units of the user space, of the specified context.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGContext.h`

Data Types

CGContextRef

An opaque type that represents a Quartz 2D drawing environment.

```
typedef struct CGContext * CGContextRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGContext.h`

Constants

Blend Modes

Compositing operations for images.

```
enum CGBlendMode {
    kCGBlendModeNormal,
    kCGBlendModeMultiply,
    kCGBlendModeScreen,
    kCGBlendModeOverlay,
    kCGBlendModeDarken,
    kCGBlendModeLighten,
    kCGBlendModeColorDodge,
    kCGBlendModeColorBurn,
    kCGBlendModeSoftLight,
    kCGBlendModeHardLight,
    kCGBlendModeDifference,
    kCGBlendModeExclusion,
    kCGBlendModeHue,
    kCGBlendModeSaturation,
    kCGBlendModeColor,
    kCGBlendModeLuminosity,
    kCGBlendModeClear,
    kCGBlendModeCopy,
    kCGBlendModeSourceIn,
    kCGBlendModeSourceOut,
    kCGBlendModeSourceAtop,
    kCGBlendModeDestinationOver,
    kCGBlendModeDestinationIn,
    kCGBlendModeDestinationOut,
    kCGBlendModeDestinationAtop,
    kCGBlendModeXOR,
    kCGBlendModePlusDarker,
    kCGBlendModePlusLighter
};
typedef enum CGBlendMode CGBlendMode;
```

Constants

`kCGBlendModeNormal`

Paints the source image samples over the background image samples.

Available in Mac OS X v10.4 and later.

Declared in `CGContext.h`.

`kCGBlendModeMultiply`

Multiplies the source image samples with the background image samples. This results in colors that are at least as dark as either of the two contributing sample colors.

Available in Mac OS X v10.4 and later.

Declared in `CGContext.h`.

`kCGBlendModeScreen`

Multiplies the inverse of the source image samples with the inverse of the background image samples. This results in colors that are at least as light as either of the two contributing sample colors.

Available in Mac OS X v10.4 and later.

Declared in `CGContext.h`.

`kCGBlendModeOverlay`
: Either multiplies or screens the source image samples with the background image samples, depending on the background color. The result is to overlay the existing image samples while preserving the highlights and shadows of the background. The background color mixes with the source image to reflect the lightness or darkness of the background.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeDarken`
: Creates the composite image samples by choosing the darker samples (either from the source image or the background). The result is that the background image samples are replaced by any source image samples that are darker. Otherwise, the background image samples are left unchanged.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeLighten`
: Creates the composite image samples by choosing the lighter samples (either from the source image or the background). The result is that the background image samples are replaced by any source image samples that are lighter. Otherwise, the background image samples are left unchanged.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeColorDodge`
: Brightens the background image samples to reflect the source image samples. Source image sample values that specify black do not produce a change.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeColorBurn`
: Darkens the background image samples to reflect the source image samples. Source image sample values that specify white do not produce a change.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeSoftLight`
: Either darkens or lightens colors, depending on the source image sample color. If the source image sample color is lighter than 50% gray, the background is lightened, similar to dodging. If the source image sample color is darker than 50% gray, the background is darkened, similar to burning. If the source image sample color is equal to 50% gray, the background is not changed. Image samples that are equal to pure black or pure white produce darker or lighter areas, but do not result in pure black or white. The overall effect is similar to what you'd achieve by shining a diffuse spotlight on the source image. Use this to add highlights to a scene.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeHardLight`
: Either multiplies or screens colors, depending on the source image sample color. If the source image sample color is lighter than 50% gray, the background is lightened, similar to screening. If the source image sample color is darker than 50% gray, the background is darkened, similar to multiplying. If the source image sample color is equal to 50% gray, the source image is not changed. Image samples that are equal to pure black or pure white result in pure black or white. The overall effect is similar to what you'd achieve by shining a harsh spotlight on the source image. Use this to add highlights to a scene.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeDifference`
: Subtracts either the source image sample color from the background image sample color, or the reverse, depending on which sample has the greater brightness value. Source image sample values that are black produce no change; white inverts the background color values.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeExclusion`
: Produces an effect similar to that produced by `kCGBlendModeDifference`, but with lower contrast. Source image sample values that are black don't produce a change; white inverts the background color values.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeHue`
: Uses the luminance and saturation values of the background with the hue of the source image.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeSaturation`
: Uses the luminance and hue values of the background with the saturation of the source image. Areas of the background that have no saturation (that is, pure gray areas) don't produce a change.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeColor`
: Uses the luminance values of the background with the hue and saturation values of the source image. This mode preserves the gray levels in the image. You can use this mode to color monochrome images or to tint color images.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeLuminosity`
: Uses the hue and saturation of the background with the luminance of the source image. This mode creates an effect that is inverse to the effect created by `kCGBlendModeColor`.

  Available in Mac OS X v10.4 and later.

  Declared in `CGContext.h`.

`kCGBlendModeClear`
`R = 0`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeCopy`
`R = S`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeSourceIn`
`R = S*Da`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeSourceOut`
`R = S*(1 - Da)`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeSourceAtop`
`R = S*Da + D*(1 - Sa)`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeDestinationOver`
`R = S*(1 - Da) + D`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeDestinationIn`
`R = D*Sa`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeDestinationOut`
`R = D*(1 - Sa)`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeDestinationAtop`
`R = S*(1 - Da) + D*Sa`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModeXOR`
`R = S*(1 - Da) + D*(1 - Sa)`. This XOR mode is only nominally related to the classical bitmap XOR operation, which is not supported by Quartz 2D.

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModePlusDarker`
`R = MAX(0, (1 - D) + (1 - S))`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

`kCGBlendModePlusLighter`
`R = MIN(1, S + D)`

Available in Mac OS X v10.5 and later.

Declared in `CGContext.h`.

Discussion
The blend mode constants introduced in Mac OS X v10.5 represent the Porter-Duff blend modes. The symbols in the equations for these blend modes are:

- R is the premultiplied result
- S is the source color, and includes alpha
- D is the destination color, and includes alpha
- Ra, Sa, and Da are the alpha components of R, S, and D

You can find more information on blend modes, including examples of images produced using them, and many mathematical descriptions of the modes, in *PDF Reference, Fourth Edition*, Version 1.5, Adobe Systems, Inc. If you are a former QuickDraw developer, it may be helpful for you to think of blend modes as an alternative to transfer modes

For examples of using blend modes see "Setting Blend Modes" and "Using Blend Modes With Images" in *Quartz 2D Programming Guide*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGContext.h`

Interpolation Qualities

Levels of interpolation quality for rendering an image.

```
enum CGInterpolationQuality {
    kCGInterpolationDefault,
    kCGInterpolationNone,
    kCGInterpolationLow,
    kCGInterpolationHigh
};
typedef enum CGInterpolationQuality CGInterpolationQuality;
```

Constants
`kCGInterpolationDefault`

The default level of quality.

Available in Mac OS X v10.1 and later.

Declared in `CGContext.h`.

`kCGInterpolationNone`
No interpolation.

Available in Mac OS X v10.1 and later.

Declared in `CGContext.h`.

`kCGInterpolationLow`
A low level of interpolation quality. This setting may speed up image rendering.

Available in Mac OS X v10.1 and later.

Declared in `CGContext.h`.

`kCGInterpolationHigh`
A high level of interpolation quality. This setting may slow down image rendering.

Available in Mac OS X v10.1 and later.

Declared in `CGContext.h`.

Discussion
You use the function `CGContextSetInterpolationQuality` (page 116) to set the interpolation quality in a graphics context.

Declared In
`CGContext.h`

Line Cap Styles

Styles for rendering the endpoint of a stroked line.

```
enum CGLineCap {
    kCGLineCapButt,
    kCGLineCapRound,
    kCGLineCapSquare
};
typedef enum CGLineCap CGLineCap;
```

Constants
`kCGLineCapButt`
A line with a squared-off end. Quartz draws the line to extend only to the exact endpoint of the path. This is the default.

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

`kCGLineCapRound`
A line with a rounded end. Quartz draws the line to extend beyond the endpoint of the path. The line ends with a semicircular arc with a radius of 1/2 the line's width, centered on the endpoint.

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

`kCGLineCapSquare`
A line with a squared-off end. Quartz extends the line beyond the endpoint of the path for a distance equal to half the line width.

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

Discussion
A line cap specifies the method used by `CGContextStrokePath` (page 134) to draw the endpoint of the line. To change the line cap style in a graphics context, you use the function `CGContextSetLineCap` (page 116).

Declared In
`CGContext.h`

Line Joins

Junction types for stroked lines.

```
enum CGLineJoin {
    kCGLineJoinMiter,
    kCGLineJoinRound,
    kCGLineJoinBevel
};
typedef enum CGLineJoin CGLineJoin;
```

Constants

`kCGLineJoinMiter`
A join with a sharp (angled) corner. Quartz draws the outer sides of the lines beyond the endpoint of the path, until they meet. If the length of the miter divided by the line width is greater than the miter limit, a bevel join is used instead. This is the default. To set the miter limit, see `CGContextSetMiterLimit` (page 119)

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

`kCGLineJoinRound`
A join with a rounded end. Quartz draws the line to extend beyond the endpoint of the path. The line ends with a semicircular arc with a radius of 1/2 the line's width, centered on the endpoint.

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

`kCGLineJoinBevel`
A join with a squared-off end. Quartz draws the line to extend beyond the endpoint of the path, for a distance of 1/2 the line's width.

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

Discussion
A line join specifies how `CGContextStrokePath` (page 134) draws the junction between connected line segments. To set the line join style in a graphics context, you use the function `CGContextSetLineJoin` (page 118).

Declared In
`CGContext.h`

Text Drawing Modes

Modes for rendering text.

```
enum CGTextDrawingMode {
    kCGTextFill,
    kCGTextStroke,
    kCGTextFillStroke,
    kCGTextInvisible,
    kCGTextFillClip,
    kCGTextStrokeClip,
    kCGTextFillStrokeClip,
    kCGTextClip
};
typedef enum CGTextDrawingMode CGTextDrawingMode;
```

Constants

`kCGTextFill`
Perform a fill operation on the text.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGTextStroke`
Perform a stroke operation on the text.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGTextFillStroke`
Perform fill, then stroke operations on the text.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGTextInvisible`
Do not draw the text, but do update the text position.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGTextFillClip`
Perform a fill operation, then intersect the text with the current clipping path.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGTextStrokeClip`
Perform a stroke operation, then intersect the text with the current clipping path.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGTextFillStrokeClip`
Perform fill then stroke operations, then intersect the text with the current clipping path.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGTextClip`
Specifies to intersect the text with the current clipping path. This mode does not paint the text.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

Discussion
You provide a text drawing mode constant to the function `CGContextSetTextDrawingMode` (page 127) to set the current text drawing mode for a graphics context. Text drawing modes determine how Quartz renders individual glyphs onscreen. For example, you can set a text drawing mode to draw text filled in or outlined (stroked) or both. You can also create special effects with the text clipping drawing modes, such as clipping an image to a glyph shape.

Declared In
`CGContext.h`

Text Encodings

Text encodings for fonts.

```
enum CGTextEncoding {
    kCGEncodingFontSpecific,
    kCGEncodingMacRoman
};
typedef enum CGTextEncoding CGTextEncoding;
```

Constants

`kCGEncodingFontSpecific`
The built-in encoding of the font.

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

`kCGEncodingMacRoman`
The MacRoman encoding. MacRoman is an ASCII variant originally created for use in the Mac OS, in which characters 127 and lower are ASCII, and characters 128 and higher are non-English characters and symbols.

Available in Mac OS X v10.0 and later.

Declared in `CGContext.h`.

Discussion
For more information on setting the font in a graphics context, see `CGContextSelectFont` (page 106).

Declared In
`CGContext.h`

CGDataConsumer Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGDataConsumer.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGDataConsumerRef` opaque type abstracts the data-writing task and eliminates the need for applications to manage data through a raw memory buffer. You can use data consumer objects to write image or PDF data and all, except for `CGDataConsumerCreateWithCFData` (page 148), are available in Mac OS X v10.0 or later.

If your application runs in Mac OS X v10.4 or later, you should use CGImageDestination objects rather than data consumers. See *CGImageDestination Reference*.

Functions by Task

Creating Data Consumers

`CGDataConsumerCreate` (page 148)
Creates a data consumer that uses callback functions to write data.

`CGDataConsumerCreateWithURL` (page 149)
Creates a data consumer that writes data to a location specified by a URL.

`CGDataConsumerCreateWithCFData` (page 148)
Creates a data consumer that writes to a CFData object.

Getting the CFType ID

`CGDataConsumerGetTypeID` (page 149)
Returns the Core Foundation type identifier for Quartz data consumers.

Retaining and Releasing Data Consumers

`CGDataConsumerRelease` (page 150)
Decrements the retain count of a data consumer.

`CGDataConsumerRetain` (page 150)
Increments the retain count of a data consumer.

Functions

CGDataConsumerCreate

Creates a data consumer that uses callback functions to write data.

```
CGDataConsumerRef CGDataConsumerCreate (
   void *info,
   const CGDataConsumerCallbacks *callbacks
);
```

Parameters

info
A pointer to data of any type or `NULL`. When Quartz calls the functions specified in the `callbacks` parameter, it passes this pointer as the `info` parameter.

callbacks
A pointer to a `CGDataConsumerCallbacks` structure that specifies the callback functions you implement to copy data sent to the consumer and to handle the consumer's basic memory management. For a complete description, see `CGDataConsumerCallbacks` (page 152).

Return Value
A new data consumer object. You are responsible for releasing this object using `CGDataConsumerRelease` (page 150).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGDataConsumer.h`

CGDataConsumerCreateWithCFData

Creates a data consumer that writes to a CFData object.

```
CGDataConsumerRef CGDataConsumerCreateWithCFData (
   CFMutableDataRef data
);
```

Parameters

data

The CFData object to write to.

Return Value

A new data consumer object. You are responsible for releasing this object using `CGDataConsumerRelease` (page 150).

Discussion

You can use this function when you need to represent Quartz data as a CFData type. For example, you might create a CFData object that you then copy to the pasteboard.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGDataConsumer.h`

CGDataConsumerCreateWithURL

Creates a data consumer that writes data to a location specified by a URL.

```
CGDataConsumerRef CGDataConsumerCreateWithURL (
   CFURLRef url
);
```

Parameters

url

A CFURL object that specifies the data destination.

Return Value

A new data consumer object. You are responsible for releasing this object using `CGDataConsumerRelease` (page 150).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGDataConsumer.h`

CGDataConsumerGetTypeID

Returns the Core Foundation type identifier for Quartz data consumers.

```
CFTypeID CGDataConsumerGetTypeID (
   void
);
```

Return Value

The Core Foundation identifier for the opaque type `CGDataConsumerRef` (page 153).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGDataConsumer.h`

CGDataConsumerRelease

Decrements the retain count of a data consumer.

```
void CGDataConsumerRelease (
   CGDataConsumerRef consumer
);
```

Parameters

consumer
The data consumer to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `consumer` parameter is `NULL`.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGDataConsumer.h`

CGDataConsumerRetain

Increments the retain count of a data consumer.

```
CGDataConsumerRef CGDataConsumerRetain (
   CGDataConsumerRef consumer
);
```

Parameters

consumer
The data consumer to retain.

Return Value
The same data consumer you passed in as the `consumer` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `consumer` parameter is `NULL`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGDataConsumer.h`

Callbacks

CGDataConsumerPutBytesCallback

Copies data from a Quartz-supplied buffer into a data consumer.

```
size_t (*CGDataConsumerPutBytesCallback) (
    void *info,
    const void *buffer,
    size_t count
);
```

If you name your function `MyConsumerPutBytes`, you would declare it like this:

```
size_t MyConsumerPutBytes (
    void *info,
    const void *buffer,
    size_t count
);
```

Parameters

info
: A generic pointer to private data shared among your callback functions. This is the pointer supplied to `CGDataConsumerCreate` (page 148).

buffer
: The Quartz-supplied buffer from which you copy the specified number of bytes.

count
: The number of bytes to copy.

Return Value
The number of bytes copied. If no more data can be written to the consumer, you should return `0`.

Discussion
When Quartz is ready to send data to the consumer, your function is called. It should copy the specified number of bytes from `buffer` into some resource under your control—for example, a file.

For information on how to associate your callback function with a data consumer, see `CGDataConsumerCreate` (page 148) and `CGDataConsumerCallbacks` (page 152).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGDataConsumer.h`

CGDataConsumerReleaseInfoCallback

Releases any private data or resources associated with the data consumer.

```
void (*CGDataConsumerReleaseInfoCallback) (
    void *info
);
```

If you name your function `MyConsumerReleaseInfo`, you would declare it like this:

```
void MyConsumerReleaseInfo (
    void *info
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataConsumerCreate` (page 148).

Discussion

When Quartz frees a data consumer that has an associated release function, the release function is called.

For information on how to associate your callback function with a data consumer, see `CGDataConsumerCreate` (page 148) and `CGDataConsumerCallbacks` (page 152).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGDataConsumer.h`

Data Types

CGDataConsumerCallbacks

A structure that contains pointers to callback functions that manage the copying of data for a data consumer.

```
struct CGDataConsumerCallbacks {
 CGDataConsumerPutBytesCallback putBytes;
 CGDataConsumerReleaseInfoCallback releaseConsumer;
};
typedef struct CGDataConsumerCallbacks CGDataConsumerCallbacks;
```

Fields

`putBytes`

A pointer to a function that copies data to the data consumer. For more information, see `CGDataConsumerPutBytesCallback` (page 151).

`releaseConsumer`

A pointer to a function that handles clean-up for the data consumer, or `NULL`. For more information, see `CGDataConsumerReleaseInfoCallback` (page 152)

Discussion
The functions specified by the `CGDataConsumerCallbacks` structure are responsible for copying data that Quartz sends to your consumer and for handling the consumer's basic memory management. You supply a `CGDataConsumerCallbacks` structure to the function `CGDataConsumerCreate` (page 148) to create a data consumer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDataConsumer.h`

CGDataConsumerRef

An opaque type that handles the storage of data supplied by Quartz functions.

```
typedef struct CGDataConsumer *CGDataConsumerRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDataConsumer.h`

CGDataProvider Reference

Derived From:	*CFType Reference*
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGDataProvider.h

Overview

The CGDataProvider header file declares a data type that supplies Quartz functions with data. Data provider objects abstract the data-access task and eliminate the need for applications to manage data through a raw memory buffer.

For information on how to use CGDataProvider functions, see *Quartz 2D Programming Guide* Programming Guide.

See also *CGDataConsumer Reference*.

Functions

CGDataProviderCopyData

Returns a copy of the provider's data.

```
CFDataRef CGDataProviderCopyData(
   CGDataProviderRef provider
);
```

Parameters

provider

The data provider whose data you want to copy.

Return Value

A new data object containing a copy of the provider's data. You are responsible for releasing this object.

Availability

Available in Mac OS X v10.5 and later.

Declared In

CGDataProvider.h

CGDataProviderCreate

Creates a Quartz sequential-access data provider. (Deprecated in Mac OS X v10.5.)

```
CGDataProviderRef CGDataProviderCreate (
   void *info,
   const CGDataProviderCallbacks *callbacks
);
```

Parameters

info

A pointer to data of any type or `NULL`. When Quartz calls the functions specified in the `callbacks` parameter, it sends each of the functions this data.

callbacks

A pointer to a `CGDataProviderCallbacks` structure that specifies the callback functions you implement to handle the data provider's basic memory management. For a complete description, see `CGDataProviderCallbacks` (page 170).

Return Value

A new data provider. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion

You use this function to create a sequential-access data provider that uses callback functions to read data from your program in a stream.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CGDataProvider.h`

CGDataProviderCreateDirect

Creates a Quartz direct-access data provider.

```
CGDataProviderRef CGDataProviderCreateDirect (
   void *info,
   off_t size,
   const CGDataProviderDirectCallbacks *callbacks
);
```

Parameters

info

A pointer to data of any type or `NULL`. When Quartz calls the functions specified in the `callbacks` parameter, it sends each of the functions this pointer.

size

The number of bytes of data to provide.

callbacks

A pointer to a `CGDataProviderDirectCallbacks` structure that specifies the callback functions you implement to handle the data provider's basic memory management.

Return Value
A new data provider. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion
You use this function to create a direct-access data provider that uses callback functions to read data from your program in a single block.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGDataProvider.h`

CGDataProviderCreateDirectAccess

Creates a Quartz direct-access data provider. (Deprecated in Mac OS X v10.5.)

```
CGDataProviderRef CGDataProviderCreateDirectAccess (
   void *info,
   size_t size,
   const CGDataProviderDirectAccessCallbacks *callbacks
);
```

Parameters

info
A pointer to data of any type or `NULL`. When Quartz calls the functions specified in the `callbacks` parameter, it sends each of the functions this pointer.

size
A value that specifies the number of bytes that the data provider contains.

callbacks
A pointer to a `CGDataProviderDirectAccessCallbacks` structure that specifies the callback functions you implement to handle the data provider's basic memory management. For a complete description, see `CGDataProviderDirectAccessCallbacks` (page 171).

Return Value
A new data provider. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion
You use this function to create a direct-access data provider that uses callback functions to read data from your program in a single block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CGDataProvider.h`

CGDataProviderCreateSequential

Creates a Quartz sequential-access data provider.

```
CGDataProviderRef CGDataProviderCreateSequential (
   void *info,
   const CGDataProviderSequentialCallbacks *callbacks
);
```

Parameters

info

A pointer to data of any type or `NULL`. When Quartz calls the functions specified in the `callbacks` parameter, it sends each of the functions this pointer.

callbacks

A pointer to a `CGDataProviderSequentialCallbacks` structure that specifies the callback functions you implement to handle the data provider's basic memory management.

Return Value

A new data provider. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion

You use this function to create a sequential-access data provider that uses callback functions to read data from your program in a single block.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CGDataProvider.h`

CGDataProviderCreateWithCFData

Creates a Quartz data provider that reads from a CFData object.

```
CGDataProviderRef CGDataProviderCreateWithCFData (
   CFDataRef data
);
```

Parameters

data

The CFData object to read from.

Return Value

A new data provider. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion

You can use this function when you need to represent Quartz data as a CFData type. For example, you might create a CFData object when reading data from the pasteboard.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGDataProvider.h`

CGDataProviderCreateWithData

Creates a Quartz direct-access data provider that uses data your program supplies.

```
CGDataProviderRef CGDataProviderCreateWithData (
   void *info,
   const void *data,
   size_t size,
   CGDataProviderReleaseDataCallback releaseData
);
```

Parameters

info

A pointer to data of any type, or `NULL`. When Quartz calls the function specified in the `releaseData` parameter, Quartz sends it this pointer as its first argument.

data

A pointer to the array of data that the provider contains.

size

A value that specifies the number of bytes that the data provider contains.

releaseData

A pointer to a release callback for the data provider, or `NULL`. Your release function is called when Quartz frees the data provider. For more information, see `CGDataProviderReleaseDataCallback` (page 166).

Return Value

A new data provider. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion

You use this function to create a direct-access data provider that uses callback functions to read data from your program an entire block at one time.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CGDataProvider.h`

CGDataProviderCreateWithFilename

Creates a Quartz direct-access data provider that uses a file to supply data.

```
CGDataProviderRef CGDataProviderCreateWithFilename(
   const char *filename
);
```

Parameters

filename

The full or relative pathname to use for the data provider. When you supply Quartz data via the provider, it reads the data from the specified file.

Return Value
A new data provider or `NULL` if the file could not be opened. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion
You use this function to create a direct-access data provider that supplies data from a file. When you supply Quartz with a direct-access data provider, Quartz obtains data from your program in a single block.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDataProvider.h`

CGDataProviderCreateWithURL

Creates a Quartz direct-access data provider that uses a URL to supply data.

```
CGDataProviderRef CGDataProviderCreateWithURL (
   CFURLRef url
);
```

Parameters

url

A CFURL object to use for the data provider. When you supply Quartz data via the provider, it reads the data from the URL address.

Return Value
A new data provider or `NULL` if the data from the URL could not be accessed. You are responsible for releasing this object using `CGDataProviderRelease` (page 161).

Discussion
You use this function to create a direct-access data provider that supplies data from a URL. When you supply Quartz with a direct-access data provider, Quartz obtains data from your program in a single entire block.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDataProvider.h`

CGDataProviderGetTypeID

Returns the Core Foundation type identifier for Quartz data providers.

```
CFTypeID CGDataProviderGetTypeID (
   void
);
```

Return Value
The identifier for the opaque type `CGDataProviderRef` (page 170).

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGDataProvider.h

CGDataProviderRelease

Decrements the retain count of a data provider.

```
void CGDataProviderRelease (
   CGDataProviderRef provider
);
```

Parameters

provider
The data provider to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `provider` parameter is `NULL`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDataProvider.h

CGDataProviderRetain

Increments the retain count of a data provider.

```
CGDataProviderRef CGDataProviderRetain (
   CGDataProviderRef provider
);
```

Parameters

provider
The data provider to retain.

Return Value
The same data provider you passed in as the `provider` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `provider` parameter is `NULL`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDataProvider.h

Callbacks by Task

Sequential-Access Data Provider Callbacks

`CGDataProviderGetBytesCallback` (page 165)
A callback function that copies from a provider data stream into a Quartz-supplied buffer.

`CGDataProviderReleaseInfoCallback` (page 167)
A callback function that releases any private data or resources associated with the data provider.

`CGDataProviderRewindCallback` (page 168)
A callback function that moves the current position in the data stream back to the beginning.

`CGDataProviderSkipBytesCallback` (page 168)
A callback function that advances the current position in the data stream supplied by the provider.

`CGDataProviderSkipForwardCallback` (page 169)
A callback function that advances the current position in the data stream supplied by the provider.

Direct-Access Data Provider Callbacks

`CGDataProviderGetBytePointerCallback` (page 162)
A callback function that returns a generic pointer to the provider data.

`CGDataProviderGetBytesAtOffsetCallback` (page 163)
A callback function that copies data from the provider into a Quartz buffer.

`CGDataProviderReleaseBytePointerCallback` (page 166)
A callback function that releases the pointer Quartz obtained by calling `CGDataProviderGetBytePointerCallback` (page 162).

`CGDataProviderReleaseDataCallback` (page 166)
A callback function that releases data you supply to the function `CGDataProviderCreateWithData` (page 159).

`CGDataProviderGetBytesAtPositionCallback` (page 164)
A callback function that copies data from the provider into a Quartz buffer.

Callbacks

CGDataProviderGetBytePointerCallback

A callback function that returns a generic pointer to the provider data.

```
const void * (*CGDataProviderGetBytePointerCallback) (
   void *info
);
```

If you name your function `MyProviderGetBytePointer`, you would declare it like this:

```
void *MyProviderGetBytePointer (
```

```
    void *info
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreateDirectAccess` (page 157).

Return Value

A generic pointer to your provider data. By suppling this pointer, you are giving Quartz read-only access to both the pointer and the underlying provider data. You must not move or modify the provider data until Quartz calls your `CGDataProviderReleaseBytePointerCallback` (page 166) function.

Discussion

When Quartz needs direct access to your provider data, this function is called.

For information on how to associate your function with a direct-access data provider, see `CGDataProviderCreateDirectAccess` (page 157) and `CGDataProviderDirectAccessCallbacks` (page 171).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGDataProvider.h`

CGDataProviderGetBytesAtOffsetCallback

A callback function that copies data from the provider into a Quartz buffer.

```
typedef size_t (*CGDataProviderGetBytesAtOffsetCallback) (
    void *info,
    void *buffer,
    size_t offset,
    size_t count
);
```

If you name your function `MyProviderGetBytesWithOffset`, you would declare it like this:

```
size_t MyProviderGetBytesWithOffset (
    void *info,
    void *buffer,
    size_t offset,
    size_t count
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreateDirectAccess` (page 157).

buffer

The Quartz-supplied buffer into which you copy the specified number of bytes.

offset
Specifies the relative location in the data provider at which to begin copying data.

count
The number of bytes to copy.

Return Value
The number of bytes copied. If no more data can be written to the buffer, you should return 0.

Discussion
When Quartz is ready to receive data from the provider, your function is called.

For information on how to associate your function with a direct-access data provider, see `CGDataProviderCreateDirectAccess` (page 157) and `CGDataProviderDirectAccessCallbacks` (page 171).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGDataProvider.h`

CGDataProviderGetBytesAtPositionCallback

A callback function that copies data from the provider into a Quartz buffer.

```
typedef size_t (*CGDataProviderGetBytesAtPositionCallback) (
    void *info,
    void *buffer,
    off_t position,
    size_t count
);
```

If you name your function `MyProviderGetBytesAtPosition`, you would declare it like this:

```
size_t MyProviderGetBytesAtPosition (
    void *info,
    void *buffer,
    off_t position,
    size_t count
);
```

Parameters

info
A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreateDirect` (page 156).

buffer
The Quartz-supplied buffer into which you copy the specified number of bytes.

position
Specifies the relative location in the data provider at which to begin copying data.

count
The number of bytes to copy.

Return Value
The number of bytes copied. If no more data can be written to the buffer, you should return 0.

Discussion
When Quartz is ready to receive data from the provider, your function is called.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGDataProvider.h`

CGDataProviderGetBytesCallback

A callback function that copies from a provider data stream into a Quartz-supplied buffer.

```
size_t (*CGDataProviderGetBytesCallback) (
    void *info,
    void *buffer,
    size_t count
);
```

If you name your function `MyProviderGetBytes`, you would declare it like this:

```
size_t MyProviderGetBytes (
    void *info,
    void *buffer,
    size_t count
);
```

Parameters

info
A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreate` (page 156).

buffer
The Quartz-supplied buffer into which you copy the specified number of bytes.

count
The number of bytes to copy.

Return Value
The number of bytes copied. If no more data can be written to the buffer, you should return `0`.

Discussion
When Quartz is ready to receive data from the provider data stream, your function is called. It should copy the specified number of bytes into `buffer`.

For information on how to associate your callback function with a data provider, see `CGDataProviderCreate` (page 156) and `CGDataProviderCallbacks` (page 170).

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGDataProvider.h

CGDataProviderReleaseBytePointerCallback

A callback function that releases the pointer Quartz obtained by calling CGDataProviderGetBytePointerCallback (page 162).

```
typedef void (*CGDataProviderReleaseBytePointerCallback) (
    void *info,
    const void *pointer
);
```

If you name your function MyProviderReleaseBytePointer, you would declare it like this:

```
void MyProviderReleaseBytePointer (
    void *info,
    const void *pointer
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to CGDataProviderCreateDirectAccess (page 157).

pointer

A pointer to your provider data. This is the same pointer you returned in CGDataProviderGetBytePointerCallback (page 162).

Discussion
When Quartz no longer needs direct access to your provider data, your function is called. You may safely modify, move, or release your provider data at this time.

For information on how to associate your function with a direct-access data provider, see CGDataProviderCreateDirectAccess (page 157) and CGDataProviderDirectAccessCallbacks (page 171).

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGDataProvider.h

CGDataProviderReleaseDataCallback

A callback function that releases data you supply to the function CGDataProviderCreateWithData (page 159).

```
typedef void (*CGDataProviderReleaseDataCallback) (
    void *info,
    const void *data
    size_t size
);
```

If you name your function `MyProviderReleaseData`, you would declare it like this:

```
void MyProviderReleaseData (
    void *info,
    const void *data
    size_t size
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreateWithData` (page 159).

data

A pointer to your provider data.

size

The size of the data.

Discussion

When Quartz no longer needs direct access to your provider data, your function is called. You may safely modify, move, or release your provider data at this time.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CGDataProvider.h`

CGDataProviderReleaseInfoCallback

A callback function that releases any private data or resources associated with the data provider.

```
void (*CGDataProviderReleaseInfoCallback) (
    void *info
);
```

If you name your function `MyProviderReleaseInfo`, you would declare it like this:

```
void MyProviderReleaseInfo (
    void *info
);
```

Parameters

info

A generic pointer to private information shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreate` (page 156).

Discussion

When Quartz frees a data provider that has an associated release function, the release function is called.

For information on how to associate your callback function with a data provider, see `CGDataProviderCreate` (page 156) and `CGDataProviderCallbacks` (page 170).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGDataProvider.h`

CGDataProviderRewindCallback

A callback function that moves the current position in the data stream back to the beginning.

```
void (*CGDataProviderRewindCallback) (
    void *info
);
```

If you name your function `MyProviderRewind`, you would declare it like this:

```
void MyProviderRewind (
    void *info
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreate` (page 156).

Discussion

When Quartz needs to read from the beginning of the provider's data stream, your function is called.

For information on how to associate your callback function with a data provider, see `CGDataProviderCreate` (page 156) and `CGDataProviderCallbacks` (page 170).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGDataProvider.h`

CGDataProviderSkipBytesCallback

A callback function that advances the current position in the data stream supplied by the provider.

```
void (*CGDataProviderSkipBytesCallback) (
    void *info,
    size_t count
);
```

If you name your function `MyProviderSkipBytes`, you would declare it like this:

```
void MyProviderSkipBytes (
    void *info,
    size_t count
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreate` (page 156).

count

The number of bytes to skip.

Discussion

When Quartz needs to advance forward in the provider's data stream, your function is called.

For information on how to associate your callback function with a data provider, see `CGDataProviderCreate` (page 156) and `CGDataProviderCallbacks` (page 170).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGDataProvider.h`

CGDataProviderSkipForwardCallback

A callback function that advances the current position in the data stream supplied by the provider.

```
off_t (*CGDataProviderSkipForwardCallback) (
    void *info,
    off_t count
);
```

If you name your function `MyProviderSkipForwardBytes`, you would declare it like this:

```
off_t MyProviderSkipForwardBytes (
    void *info,
    off_t count
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGDataProviderCreate` (page 156).

count
The number of bytes to skip.

Return Value
The number of bytes that were actually skipped.

Discussion
When Quartz needs to advance forward in the provider's data stream, your function is called.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGDataProvider.h`

Data Types

CGDataProviderRef

Defines an opaque type that supplies Quartz with data.

```
typedef struct CGDataProvider *CGDataProviderRef;
```

Discussion
Some Quartz routines supply blocks of data to your program. Rather than reading through a raw memory buffer, data provider objects of type `CGDataProviderRef` allow you to supply Quartz functions with data.

In Mac OS X version 10.2 and later, `CGDataProviderRef` is derived from `CFTypeRef` and inherits the properties that all Core Foundation types have in common. For more information, see *CFType Reference*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDataProvider.h`

CGDataProviderCallbacks

Defines a structure containing pointers to client-defined callback functions that manage the sending of data for a sequential-access data provider.

```
struct CGDataProviderCallbacks {
    CGDataProviderGetBytesCallback getBytes;
    CGDataProviderSkipBytesCallback skipBytes;
    CGDataProviderRewindCallback rewind;
    CGDataProviderReleaseInfoCallback releaseProvider;
};
typedef struct CGDataProviderCallbacks CGDataProviderCallbacks;
```

Fields

`getBytes`

A pointer to a function that copies data from the provider. For more information, see `CGDataProviderGetBytesCallback` (page 165).

`skipBytes`

A pointer to a function that Quartz calls to advance the stream of data supplied by the provider. For more information, see `CGDataProviderSkipBytesCallback` (page 168).

`rewind`

A pointer to a function Quartz calls to return the provider to the beginning of the data stream. For more information, see `CGDataProviderRewindCallback` (page 168).

`releaseProvider`

A pointer to a function that handles clean-up for the data provider, or `NULL`. For more information, see `CGDataProviderReleaseInfoCallback` (page 167).

Discussion

The functions specified by the `CGDataProviderCallbacks` structure are responsible for sequentially copying data to a memory buffer for Quartz to use. The functions are also responsible for handling the data provider's basic memory management. You supply a `CGDataProviderCallbacks` structure to the function `CGDataProviderCreate` (page 156) to create a sequential-access data provider.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDataProvider.h`

CGDataProviderDirectAccessCallbacks

Defines pointers to client-defined callback functions that manage the sending of data for a direct-access data provider.

```
struct CGDataProviderDirectAccessCallbacks {
    CGDataProviderGetBytePointerCallback getBytePointer;
    CGDataProviderReleaseBytePointerCallback releaseBytePointer;
    CGDataProviderGetBytesAtOffsetCallback getBytes;
    CGDataProviderReleaseInfoCallback releaseProvider;
};
typedef struct CGDataProviderDirectAccessCallbacks
CGDataProviderDirectAccessCallbacks;
```

Fields

`getBytePointer`

A pointer to a function that returns a pointer to the provider's data. For more information, see `CGDataProviderGetBytePointerCallback` (page 162).

releaseBytePointer
: A pointer to a function that Quartz calls to release a pointer to the provider's data. For more information, see CGDataProviderReleaseBytePointerCallback (page 166).

getBytes
: A pointer to a function that copies data from the provider. For more information, see CGDataProviderGetBytesAtOffsetCallback (page 163).

releaseProvider
: A pointer to a function that handles clean-up for the data provider, or NULL. For more information, see CGDataProviderReleaseInfoCallback (page 167).

Discussion

You supply a CGDataProviderDirectAccessCallbacks structure to the function CGDataProviderCreateDirectAccess (page 157) to create a data provider for direct access. The functions specified by the CGDataProviderDirectAccessCallbacks structure are responsible for copying data a block at a time to a memory buffer for Quartz to use. The functions are also responsible for handling the data provider's basic memory management. For the callback to work, one of the getBytePointer and getBytes parameters must be non-NULL. If both are non-NULL, then getBytePointer is used to access the data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CGDataProvider.h

CGDataProviderDirectCallbacks

Defines pointers to client-defined callback functions that manage the sending of data for a direct-access data provider.

```
struct CGDataProviderDirectCallbacks {
     unsigned int version;
    CGDataProviderGetBytePointerCallback getBytePointer;
    CGDataProviderReleaseBytePointerCallback releaseBytePointer;
    CGDataProviderGetBytesAtPositionCallback getBytesAtPosition;
    CGDataProviderReleaseInfoCallback releaseInfo;
};
typedef struct CGDataProviderDirectCallbacks  CGDataProviderDirectCallbacks;
```

Fields

version
: The version of this structure. It should be set to 0.

getBytePointer
: A pointer to a function that returns a pointer to the provider's data. For more information, see CGDataProviderGetBytePointerCallback (page 162).

releaseBytePointer
: A pointer to a function that Quartz calls to release a pointer to the provider's data. For more information, see CGDataProviderReleaseBytePointerCallback (page 166).

getBytesAtPosition
: A pointer to a function that copies data from the provider.

releaseInfo
A pointer to a function that handles clean-up for the data provider, or NULL. For more information, see CGDataProviderReleaseInfoCallback (page 167).

Discussion
You supply a CGDataProviderDirectCallbacks structure to the function CGDataProviderCreateDirect (page 156) to create a data provider for direct access. The functions specified by the CGDataProviderDirectCallbacks structure are responsible for copying data a block at a time to a memory buffer for Quartz to use. The functions are also responsible for handling the data provider's basic memory management. For the callback to work, one of the getBytePointer and getBytesAtPosition parameters must be non-NULL. If both are non-NULL, then getBytePointer is used to access the data.

Availability
Available in Mac OS X v10.5 and later.

Declared In
CGDataProvider.h

CGDataProviderSequentialCallbacks

Defines a structure containing pointers to client-defined callback functions that manage the sending of data for a sequential-access data provider.

```
struct CGDataProviderSequentialCallbacks {
    unsigned int version;
    CGDataProviderGetBytesCallback getBytes;
    CGDataProviderSkipForwardCallback skipForward;
    CGDataProviderRewindCallback rewind;
    CGDataProviderReleaseInfoCallback releaseInfo;
};
typedef struct CGDataProviderSequentialCallbacks CGDataProviderSequentialCallbacks;
```

Fields
version
The version of this structure. It should be set to 0.

getBytes
A pointer to a function that copies data from the provider. For more information, see CGDataProviderGetBytesCallback (page 165).

skipForward
A pointer to a function that Quartz calls to advance the stream of data supplied by the provider.

rewind
A pointer to a function Quartz calls to return the provider to the beginning of the data stream. For more information, see CGDataProviderRewindCallback (page 168).

releaseInfo
A pointer to a function that handles clean-up for the data provider, or NULL. For more information, see CGDataProviderReleaseInfoCallback (page 167).

Discussion
The functions specified by the CGDataProviderSequentialCallbacks structure are responsible for sequentially copying data to a memory buffer for Quartz to use. The functions are also responsible for handling the data provider's basic memory management. You supply a CGDataProviderCallbacks structure to the function CGDataProviderCreateSequential (page 157) to create a sequential-access data provider.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGDataProvider.h`

CGFont Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGFont.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGFontRef` opaque type encapsulates font information. A font is a set of shapes or glyphs associated with a character set. A glyph can represent a single character (such as 'b'), more than one character (such as the "fi" ligature), or a special character such as a space. Quartz retrieves the glyphs for the font from ATS (Apple Type Services) and paints the glyphs based on the relevant parameters of the current graphics state.

Quartz provides a limited, low-level interface for drawing text. For information on text-drawing functions, see *CGContext Reference*. For full Unicode and text-layout support, use the services provided by Core Text or ATSUI).

Functions by Task

Retaining and Releasing a CGFont Object

`CGFontRelease` (page 190)
Decrements the retain count of a Quartz font.

`CGFontRetain` (page 191)
Increments the retain count of a Quartz font.

Creating a CGFont Object

`CGFontCreateWithDataProvider` (page 182)
Creates a font object from data supplied from a data provider.

`CGFontCreateWithFontName` (page 183)
Creates a font object corresponding to the font specified by a PostScript or full name.

`CGFontCreateCopyWithVariations` (page 181)
Creates a copy of a font using a variation specification dictionary.

`CGFontCreateWithPlatformFont` (page 183)
Creates a font object from an Apple Type Services (ATS) font.

Working With PostScript Fonts

`CGFontCopyPostScriptName` (page 178)
Obtains the PostScript name of a font.

`CGFontCanCreatePostScriptSubset` (page 177)
Determines whether Quartz can create a subset of the font in PostScript format.

`CGFontCreatePostScriptSubset` (page 182)
Creates a subset of the font in the specified PostScript format.

`CGFontCreatePostScriptEncoding` (page 181)
Creates a PostScript encoding of a font.

Working With Font Tables

`CGFontCopyTableTags` (page 179)
Returns an array of tags that correspond to the font tables for a font.

`CGFontCopyTableForTag` (page 179)
Returns the font table that corresponds to the provided tag.

Getting Font Information

`CGFontGetTypeID` (page 189)
Returns the Core Foundation type identifier for Quartz fonts.

`CGFontCopyVariationAxes` (page 180)
Returns an array of the variation axis dictionaries for a font.

`CGFontCopyVariations` (page 180)
Returns the variation specification dictionary for a font.

`CGFontCopyFullName` (page 177)
Returns the full name associated with a font object.

`CGFontGetAscent` (page 184)
Returns the ascent of a font.

`CGFontGetDescent` (page 185)
Returns the descent of a font.

`CGFontGetLeading` (page 188)
Returns the leading of a font.

`CGFontGetCapHeight` (page 184)
Returns the cap height of a font.

`CGFontGetXHeight` (page 190)
Returns the x-height of a font.

`CGFontGetFontBBox` (page 185)
Returns the bounding box of a font.

`CGFontGetItalicAngle` (page 188)
Returns the italic angle of a font.

`CGFontGetStemV` (page 189)
Returns the thickness of the dominant vertical stems of glyphs in a font.

`CGFontGetGlyphBBoxes` (page 187)
Get the bounding box of each glyph in an array.

`CGFontGetGlyphWithGlyphName` (page 187)
Returns the glyph for the font name associated with the specified font object.

`CGFontCopyGlyphNameForGlyph` (page 178)
Returns the glyph name associated with a font object.

`CGFontGetNumberOfGlyphs` (page 188)
Returns the number of glyphs in a font.

`CGFontGetGlyphAdvances` (page 186)
Gets the bound box of each glyph in the provided array.

`CGFontGetUnitsPerEm` (page 190) Deprecated in Mac OS X v10.5
Returns the number of glyph space units per em for the provided font.

Functions

CGFontCanCreatePostScriptSubset

Determines whether Quartz can create a subset of the font in PostScript format.

```
bool CGFontCanCreatePostScriptSubset (
   CGFontRef font,
   CGFontPostScriptFormat format
);
```

Parameters

font
A font object.

Return Value

Returns `true` if a subset in the PostScript format can be created for the font; `false` otherwise.

Discussion

For more information on PostScript format, see *Adobe Type 1 Font Format*, which is available from http://partners.adobe.com/.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGFont.h`

CGFontCopyFullName

Returns the full name associated with a font object.

```
CFStringRef CGFontCopyFullName (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value
The full name associated with the font.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
CGFont.h

CGFontCopyGlyphNameForGlyph

Returns the glyph name associated with a font object.

```
CFStringRef CGFontCopyGlyphNameForGlyph (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value
A glyph name, or `NULL` if there isn't a glyph associated with the font object.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
CGFont.h

CGFontCopyPostScriptName

Obtains the PostScript name of a font.

```
CFStringRef CGFontCopyPostScriptName (
   CGFontRef font
);
```

Parameters

font

A font object.

Return Value
The PostScript name of the font.

Discussion

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGFont.h`

CGFontCopyTableForTag

Returns the font table that corresponds to the provided tag.

```
CFDataRef CGFontCopyTableForTag(
    CGFontRef font,
    uint32_t tag
);
```

Parameters

font
A font object.

tag
The tag for the table you want to obtain.

Return Value
The font table that corresponds to the tag, or `NULL` if no such table exists.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGFont.h`

CGFontCopyTableTags

Returns an array of tags that correspond to the font tables for a font.

```
CFArrayRef CGFontCopyTableTags(
    CGFontRef font
);
```

Parameters

font
A CGFont object.

Return Value
An array of font table tags.

Discussion
Each entry in the returned array is a four-byte value that represents a single TrueType or OpenType font table tag. To obtain a tag at index `k` in a manner that is appropriate for 32-bit and 64-bit architectures, you need to use code similar to the following:

```
tag = (uint32_t)(uintptr_t)CFArrayGetValue(table, k);
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGFont.h`

CGFontCopyVariationAxes

Returns an array of the variation axis dictionaries for a font.

```
CFArrayRef CGFontCopyVariationAxes (
   CGFontRef font
);
```

Parameters

font
A CGFont object.

Return Value
An array of the variation axis dictionaries. Returns `NULL` if the font doesn't support variations.

Discussion
A variation axis is a range included in a font by the font designer that allows a font to produce different type styles. Each variation axis dictionary contains key-value pairs that specify the variation axis name and the minimum, maximum, and default values for that variation axis.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGFont.h`

CGFontCopyVariations

Returns the variation specification dictionary for a font.

```
CFDictionaryRef CGFontCopyVariations (
   CGFontRef font
);
```

Parameters

font
A font object.

Return Value
The variation specification dictionary for the font. Returns `NULL` if the font doesn't support variations.

Discussion
The variation specification dictionary contains keys that correspond to the variation axis names of the font. Each key is a variation axis name. The value for each key is the value specified for that particular variation axis represented as a CFNumber object.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGFont.h

CGFontCreateCopyWithVariations

Creates a copy of a font using a variation specification dictionary.

```
CGFontRef CGFontCreateCopyWithVariations (
   CGFontRef font,
   CFDictionaryRef variations
);
```

Parameters

font

The Quartz font to copy.

variations

A variation specification dictionary that contains keys corresponding to the variation axis names of the font. Each key in the dictionary is a variation axis name. The value for each key is the value specified for that particular variation axis represented as a CFNumber object. If a variation axis name is not specified in `variations`, then the current value from `font` is used.

Return Value
The font object.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGFont.h

CGFontCreatePostScriptEncoding

Creates a PostScript encoding of a font.

```
CFDataRef CGFontCreatePostScriptEncoding (
   CGFontRef font,
   const CGGlyph encoding[256]
);
```

Parameters

font

A CGFont object.

encoding

The encoding to use.

Return Value
A PostScript encoding of the font that contains glyphs in the specified encoding.

Discussion
For more information on PostScript format, see *Adobe Type 1 Font Format*, which is available from http://partners.adobe.com/.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGFont.h`

CGFontCreatePostScriptSubset

Creates a subset of the font in the specified PostScript format.

```
CFDataRef CGFontCreatePostScriptSubset (
   CGFontRef font,
   CFStringRef subsetName,
   CGFontPostScriptFormat format,
   const CGGlyph glyphs[],
   size_t count,
   const CGGlyph encoding[256]
);
```

Parameters

font
A font object.

subsetName
The name of the subset.

format
The PostScript format of the font.

glyphs
An array that contains the glyphs in the subset.

count
The number of glyphs specified by the `glyphs` array.

encoding
The default encoding for the subset. You can pass `NULL` if you do not want to specify an encoding.

Return Value
A subset of the font created from the supplied parameters.

Discussion
For more information on PostScript format, see *Adobe Type 1 Font Format*, which is available from http://partners.adobe.com/.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGFont.h`

CGFontCreateWithDataProvider

Creates a font object from data supplied from a data provider.

```
CGFontRef CGFontCreateWithDataProvider (
    CGDataProviderRef provider
);
```

Parameters

provider

A data provider.

Return Value

The font object or `NULL` if the font can't be created. You are responsible for releasing this object using `CGFontRelease` (page 190).

Discussion

Before drawing text in a Quartz context, you must set the font in the current graphics state by calling the function `CGContextSetFontSize` (page 114).

Availability

Available in Mac OS X version 10.5 and later.

Declared In

`CGFont.h`

CGFontCreateWithFontName

Creates a font object corresponding to the font specified by a PostScript or full name.

```
CGFontRef CGFontCreateWithFontName (
    CFStringRef name
);
```

Parameters

name

The PostScript or full name of a font.

Return Value

The font object or `NULL` if the font can't be created. You are responsible for releasing this object using `CGFontRelease` (page 190).

Discussion

Before drawing text in a Quartz context, you must set the font in the current graphics state by calling the function `CGContextSetFont` (page 113).

Availability

Available in Mac OS X version 10.5 and later.

Declared In

`CGFont.h`

CGFontCreateWithPlatformFont

Creates a font object from an Apple Type Services (ATS) font.

```
CGFontRef CGFontCreateWithPlatformFont (
   void *platformFontReference
);
```

Parameters

platformFontReference

A generic pointer to a font object. The font should be of a type appropriate to the platform on which your program is running. For Mac OS X, you should pass a pointer to an ATS font.

Return Value

The font object, or `NULL` if the platform font could not be located. You are responsible for releasing this object using `CGFontRelease` (page 190).

Discussion

Before drawing text in a Quartz context, you must set the font in the current graphics state. For ATS Fonts, call this function to create a Quartz font, and pass it to `CGContextSetFont` (page 113).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGFont.h`

CGFontGetAscent

Returns the ascent of a font.

```
int CGFontGetAscent (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value

The ascent of the font.

Discussion

The ascent is the maximum distance above the baseline of glyphs in a font. The value is specified in glyph space units.

Availability

Available in Mac OS X version 10.5 and later.

Declared In

`CGFont.h`

CGFontGetCapHeight

Returns the cap height of a font.

```
int CGFontGetCapHeight (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value
The cap height of the font.

Discussion
The cap height is the distance above the baseline of the top of flat capital letters of glyphs in a font. The value is specified in glyph space units.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CGFont.h`

CGFontGetDescent

Returns the descent of a font.

```
int CGFontGetDescent (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value
The descent of the font .

Discussion
The descent is the maximum distance below the baseline of glyphs in a font. The value is specified in glyph space units.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CGFont.h`

CGFontGetFontBBox

Returns the bounding box of a font.

```
CGRect CGFontGetFontBBox (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value
The bounding box of the font.

Discussion
The font bounding box is the union of all of the bounding boxes for all the glyphs in a font. The value is specified in glyph space units.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CGFont.h`

CGFontGetGlyphAdvances

Gets the bound box of each glyph in the provided array.

```
bool CGFontGetGlyphAdvances (
    CGFontRef font,
    const CGGlyph glyphs[],
    size_t count,
    int advances[]
);
```

Parameters

font

The font object associated with the provided glyphs.

glyphs

An array of glyphs.

count

The number of glyphs in the array.

advances

On output, an array of of advances for the provided glyphs.

Return Value
`TRUE` unless the advances can't be provided for some reason.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGFont.h`

CGFontGetGlyphBBoxes

Get the bounding box of each glyph in an array.

```
bool CGFontGetGlyphBBoxes (
    CGFontRef font,
    const CGGlyph glyphs[],
    size_t count,
    CGRect bboxes[]
);
```

Parameters

font

A font object.

glyphs

A array of glyphs.

count

The number of items in the `glyphs` array.

bboxes

On return, the bounding boxes for each glyph.

Return Value

`false` if bounding boxes can't be retrieved for any reason; `true` otherwise.

Availability

Available in Mac OS X version 10.5 and later.

Declared In

`CGFont.h`

CGFontGetGlyphWithGlyphName

Returns the glyph for the font name associated with the specified font object.

```
CGGlyph CGFontGetGlyphWithGlyphName (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value

A glyph, or `0` if there isn't a name associated with the font object.

Availability

Available in Mac OS X version 10.5 and later.

Declared In

`CGFont.h`

CGFontGetItalicAngle

Returns the italic angle of a font.

```
CGFloat CGFontGetItalicAngle (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value

The italic angle of the font, measured in degrees counter-clockwise from the vertical.

Availability

Available in Mac OS X version 10.5 and later.

Declared In

CGFont.h

CGFontGetLeading

Returns the leading of a font.

```
int CGFontGetLeading (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value

The leading of the font.

Discussion

The leading is the spacing between consecutive lines of text in a font. The value is specified in glyph space units.

Availability

Available in Mac OS X version 10.5 and later.

Declared In

CGFont.h

CGFontGetNumberOfGlyphs

Returns the number of glyphs in a font.

```
size_t CGFontGetNumberOfGlyphs (
    CGFontRef font
);
```

Parameters

font

A `CGFont` object.

Return Value
The number of glyphs in the provided font.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGFont.h`

CGFontGetStemV

Returns the thickness of the dominant vertical stems of glyphs in a font.

```
CGFloat CGFontGetItalicAngle (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value
The thickness of the dominant vertical stems of glyphs in a font.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CGFont.h`

CGFontGetTypeID

Returns the Core Foundation type identifier for Quartz fonts.

```
CFTypeID CGFontGetTypeID (
   void
);
```

Return Value
The Core Foundation identifier for the opaque type `CGFontRef` (page 191).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGFont.h`

CGFontGetUnitsPerEm

Returns the number of glyph space units per em for the provided font.

```
int CGFontGetUnitsPerEm (
    CGFontRef font
);
```

Parameters

font

A `CGFont` object.

Return Value
The number of glyph space units per em for the provided font.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGFont.h`

CGFontGetXHeight

Returns the x-height of a font.

```
int CGFontGetXHeight (
    CGFontRef font
);
```

Parameters

font

A font object.

Return Value
The x-height of the font.

Discussion
The x-height is the distance above the baseline of the top of flat, non-ascending lowercase letters (such as `x`) of glyphs in a font. The value is specified in glyph space units.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CGFont.h`

CGFontRelease

Decrements the retain count of a Quartz font.

```
void CGFontRelease (
   CGFontRef font
);
```

Parameters

font

The Quartz font to release.

Discussion

This function is equivalent to `CFRelease`, except that it does not cause an error if the `font` parameter is `NULL`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGFont.h`

CGFontRetain

Increments the retain count of a Quartz font.

```
CGFontRef CGFontRetain (
   CGFontRef font
);
```

Parameters

font

The Quartz font to retain.

Return Value

The same font you specified in the `font` parameter.

Discussion

This function is equivalent to `CFRetain`, except that it does not cause an error if the `font` parameter is `NULL`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGFont.h`

Data Types

CGFontRef

An opaque type that encapsulates font information.

```
typedef struct CGFont *CGFontRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`CGFont.h`

CGFontIndex

An index into a font table.

```
typedef unsigned short CGFontIndex;
```

Discussion
This integer type provides an additional way to specify a glyph identifier. `CGFontIndex` is equivalent to `CGGlyph` (page 192), and you can use constants of either type interchangeably.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGFont.h`

CGGlyph

An index into the internal glyph table of a font.

```
typedef unsigned short CGGlyph;
```

Discussion
When drawing text, you typically specify a sequence of characters. However, Quartz also allows you to use `CGGlyph` values to specify glyphs. In either case, Quartz renders the text using font data provided by the Apple Type Services (ATS) framework.

You provide `CGGlyph` values to the functions `CGContextShowGlyphs` (page 129) and `CGContextShowGlyphsAtPoint` (page 129). These functions display an array of glyphs at the current text position or at a position you specify, respectively.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGFont.h`

Constants

CGFontPostScriptFormat

Possible formats for a PostScript font subset.

```
enum CGFontPostScriptFormat {
    kCGFontPostScriptFormatType1 = 1,
    kCGFontPostScriptFormatType3 = 3,
    kCGFontPostScriptFormatType42 = 42
};
typedef enum CGFontPostScriptFormat CGFontPostScriptFormat;
```

Constants

`kCGFontPostScriptFormatType1`

This is documented in *Adobe Type 1 Font Format*, which is available from http://partners.adobe.com/.

Available in Mac OS X v10.4 and later.

Declared in `CGFont.h`.

`kCGFontPostScriptFormatType3`

This is documented in *PostScript Language Reference, 3rd edition*, which is available from http://partners.adobe.com/.

Available in Mac OS X v10.4 and later.

Declared in `CGFont.h`.

`kCGFontPostScriptFormatType42`

This is documented in *Adobe Technical Note 5012, The Type 42 Font Format Specification*, which is available from http://partners.adobe.com/.

Available in Mac OS X v10.4 and later.

Declared in `CGFont.h`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGFont.h`

Font Table Index Values

Possible values for an index into a font table.

```
enum {
    kCGFontIndexMax = ((1 << 16) - 2),
    kCGFontIndexInvalid = ((1 << 16) - 1),
    kCGGlyphMax = kCGFontIndexMax
};
```

Constants

`kCGFontIndexMax`

The maximum allowed value for `CGFontIndex` (page 192).

Available in Mac OS X v10.1 and later.

Declared in `CGFont.h`.

`kCGFontIndexInvalid`

An invalid font index (a value which never represents a valid glyph).

Available in Mac OS X v10.1 and later.

Declared in `CGFont.h`.

`kCGGlyphMax`
The same as `kCGFontIndexMax`.

Available in Mac OS X v10.1 and later.

Declared in `CGFont.h`.

Discussion
See `CGFontIndex` (page 192).

Declared In
`CGFont.h`

Font Variation Axis Keys

Keys used for a font variation axis dictionary.

```
const CFStringRef kCGFontVariationAxisName
const CFStringRef kCGFontVariationAxisMinValue
const CFStringRef kCGFontVariationAxisMaxValue
const CFStringRef kCGFontVariationAxisDefaultValue
```

Constants
`kCGFontVariationAxisName`
The key used to obtain the variation axis name from a variation axis dictionary. The value obtained with this key is a `CFStringRef` that specifies the name of the variation axis.

Available in Mac OS X v10.4 and later.

Declared in `CGFont.h`.

`kCGFontVariationAxisMinValue`
The key used to obtain the minimum variation axis value from a variation axis dictionary. The value obtained with this key is a `CFNumberRef` that specifies the minimum value of the variation axis.

Available in Mac OS X v10.4 and later.

Declared in `CGFont.h`.

`kCGFontVariationAxisMaxValue`
The key used to obtain the maximum variation axis value from a variation axis dictionary. The value obtained with this key is a `CFNumberRef` that specifies the maximum value of the variation axis.

Available in Mac OS X v10.4 and later.

Declared in `CGFont.h`.

`kCGFontVariationAxisDefaultValue`
The key used to obtain the default variation axis value from a variation axis dictionary. The value obtained with this key is a `CFNumberRef` that specifies the default value of the variation axis.

Available in Mac OS X v10.4 and later.

Declared in `CGFont.h`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGFont.h`

CGFunction Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGFunction.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGFunctionRef` opaque type provides a general facility for defining and using callback functions. These functions can take an arbitrary number of floating-point input values and pass back an arbitrary number of floating-point output values.

Quartz uses CGFunction objects to implement shadings. *CGShading Reference* describes the parameters and semantics required for the callbacks used by CGFunction objects.

Functions by Task

Creating a CGFunction Object

`CGFunctionCreate` (page 196)
Creates a Quartz function.

Retaining and Releasing CGFunction Objects

`CGFunctionRelease` (page 197)
Decrements the retain count of a function object.

`CGFunctionRetain` (page 197)
Increments the retain count of a function object.

Getting the CFType ID

`CGFunctionGetTypeID` (page 197)
Returns the type identifier for Quartz function objects.

Functions

CGFunctionCreate

Creates a Quartz function.

```
CGFunctionRef CGFunctionCreate (
   void *info,
   size_t domainDimension,
   const CGFloat *domain,
   size_t rangeDimension,
   const CGFloat *range,
   const CGFunctionCallbacks *callbacks
);
```

Parameters

info

A pointer to user-defined storage for data that you want to pass to your callbacks. You need to make sure that the data persists for as long as it's needed, which can be beyond the scope in which the Quartz function is used.

domainDimension

The number of inputs.

domain

An array of (`2*domainDimension`) floats used to specify the valid intervals of input values. For each `k` from `0` to `(domainDimension - 1)`, `domain[2*k]` must be less than or equal to `domain[2*k+1]`, and the `k`th input value will be clipped to lie in the interval `domain[2*k] ≤ input[k] ≤ domain[2*k+1]`. If this parameter is `NULL`, then the input values are not clipped.

rangeDimension

The number of outputs.

range

An array of `(2*rangeDimension)` floats that specifies the valid intervals of output values. For each `k` from `0` to `(rangeDimension - 1)`, `range[2*k]` must be less than or equal to `range[2*k+1]`, and the `k`th output value will be clipped to lie in the interval `range[2*k] ≤ output[k] ≤ range[2*k+1]`. If this parameter is `NULL`, then the output values are not clipped.

callbacks

A pointer to a callback function table. This table should contain pointers to the callbacks you provide to implement the semantics of this Quartz function. Quartz makes a copy of your table, so, for example, you could safely pass in a pointer to a structure on the stack.

Return Value

The new Quartz function. You are responsible for releasing this object using `CGFunctionRelease` (page 197).

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGFunction.h`

CGFunctionGetTypeID

Returns the type identifier for Quartz function objects.

```
CFTypeID CGFunctionGetTypeID (
    void
);
```

Return Value
The identifier for the opaque type `CGFunctionRef` (page 199).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGFunction.h`

CGFunctionRelease

Decrements the retain count of a function object.

```
void CGFunctionRelease (
   CGFunctionRef function
);
```

Parameters

`function`
The function object to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `function` parameter is `NULL`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGFunction.h`

CGFunctionRetain

Increments the retain count of a function object.

```
CGFunctionRef CGFunctionRetain (
   CGFunctionRef function
);
```

Parameters

`function`
The same function object you passed in as the `function` parameter.

Return Value

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `function` parameter is `NULL`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGFunction.h`

Callbacks

CGFunctionEvaluateCallback

Performs custom operations on the supplied input data to produce output data.

```
typedef void (*CGFunctionEvaluateCallback) (
    void *info,
    const float *inData,
    float *outData
);
```

If you name your function `MyCGFunctionEvaluate`, you would declare it like this:

```
void MyCGFunctionEvaluate (
    void *info,
    const float *inData,
    float *outData
);
```

Parameters

info

The `info` parameter passed to `CGFunctionCreate` (page 196).

inData

An array of floats. The size of the array is that specified by the `domainDimension` parameter passed to the `CGFunctionCreate` (page 196) function.

outData

An array of floats. The size of the array is that specified by the `rangeDimension` parameter passed to the `CGFunctionCreate` (page 196) function.

Discussion
The callback you write is responsible for implementing the calculation of output values from the supplied input values. For example, if you want to implement a simple "squaring" function of one input argument to one output argument, your evaluation function might be:

```
void evaluateSquare(void *info, const float *inData, float *outData)
{
    outData[0] = inData[0] * inData[0];
}
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGFunction.h

CGFunctionReleaseInfoCallback

Performs custom clean-up tasks when Quartz deallocates a CGFunction object.

```
typedef void (*CGFunctionReleaseInfoCallback) (
    void *info
);
```

If you name your function `MyCGFunctionReleaseInfo`, you would declare it like this:

```
void MyCGFunctionReleaseInfo (
    void *info
);
```

Parameters

info

The `info` parameter passed to `CGFunctionCreate` (page 196).

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGFunction.h

Data Types

CGFunctionRef

An opaque type that represents a callback function.

```
typedef struct CGFunction *CGFunctionRef;
```

Availability
Available in Mac OS X version 10.2 and later.

Declared In
CGFunction.h

CGFunctionCallbacks

A structure that contains callbacks needed by a CGFunction object.

```
struct CGFunctionCallbacks
{
  unsigned int version;
  CGFunctionEvaluateCallback evaluate;
  CGFunctionReleaseInfoCallback releaseInfo
};

typedef struct CGFunctionCallbacks CGFunctionCallbacks;
```

Fields

`version`

The structure version number. For this structure, the version should be `0`.

`evaluate`

The callback that evaluates the function.

`releaseInfo`

If non-`NULL`, the callback used to release the `info` parameter passed to `CGFunctionCreate` (page 196).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGFunction.h`

CGGLContext Reference

Derived From:	`CGContextRef` (page 137)
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGGLContext.h
Companion guide	Quartz 2D Programming Guide

Overview

The CGGLContext header file defines functions that create and update a graphics context for OpenGL drawing. A CGGLContext context is a type of `CGContextRef` (page 137) that is used for OpenGL content. However, its use is not recommended.

Functions

CGGLContextCreate

Creates a Quartz graphics context from an OpenGL context.

```
CGContextRef CGGLContextCreate (
   void *glContext,
   CGSize size,
   CGColorSpaceRef colorspace
);
```

Parameters

glContext

The context that the OpenGL system uses to manage OpenGL drawing.

size

The dimensions of the OpenGL viewport rectangle.

colorspace

An RGB color space that serves as the destination space when rendering device-independent colors. If `NULL`, Quartz uses the default RGB color space. Quartz retains the color space you pass in; on return, you may safely release it.

Return Value

A new Quartz graphics context. You are responsible for releasing this object by calling `CGContextRelease` (page 102).

Discussion
The use of this function is not recommended.

Creates a Quartz context from the OpenGL context `glContext`. The context establishes an OpenGL viewport rectangle with dimensions specified by the `size` parameter by calling `glViewport(3G)`. If non-`NULL`, the `colorspace` parameter should be an RGB profile that specifies the destination space when rendering device-independent colors.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGGLContext.h`

CGGLContextUpdateViewportSize

Updates the size of the viewport associated with an OpenGL context.

```
void CGGLContextUpdateViewportSize (
   CGContextRef c,
   CGSize size
);
```

Parameters

context
A Quartz graphics context obtained by calling `CGGLContextCreate` (page 201).

size
The new dimensions of the OpenGL viewport.

Discussion
The use of this function is not recommended.

You should call this function whenever the size of the associated OpenGL context changes.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGGLContext.h`

CGGradient Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGGradient.h
Companion guide	Quartz 2D Programming Guide

Overview

A gradient defines a smooth transition between colors across an area. The `CGGradientRef` opaque type, and the functions that operate on it, make creating and using radial and axial gradient fills an easy task. A CGGradient object has a color space, two or more colors, and a location for each color. The color space cannot be a pattern or indexed color space, otherwise it can be any Quartz color space (`CGColorSpaceRef` (page 56)).

Colors can be provided as component values (such as red, green, blue) or as Quartz color objects (`CGColorRef` (page 41)). In Quartz, component can vary from 0.0 to 1.0, designating the proportion of the component present in the color.

A location is a normalized value. When it comes time to paint the gradient, Quartz maps the normalized location values to the points in coordinate space that you provide.

If you want more precise control over gradients, or if your application runs in versions of Mac OS X that are earlier than v10.5, see *CGShading Reference*.

Functions by Task

Creating a CGGradient Object

`CGGradientCreateWithColorComponents` (page 204)
Creates a CGGradient object from a color space and the provided color components and locations.

`CGGradientCreateWithColors` (page 205)
Creates a CGGradient object from a color space and the provided color objects and locations.

Retaining and Releasing a CGGradient Object

`CGGradientRelease` (page 206)
Decrements the retain count of a CGGradient object.

`CGGradientRetain` (page 206)
Increments the retain count of a CGGradient object.

Getting the Type ID for a CGGradient Object

`CGGradientGetTypeID` (page 206)
Returns the Core Foundation type identifier for CGGradient objects.

Functions

CGGradientCreateWithColorComponents

Creates a CGGradient object from a color space and the provided color components and locations.

```
CGGradientRef CGGradientCreateWithColorComponents(
    CGColorSpaceRef space,
    const CGFloat components[],
    const CGFloat locations[],
    size_t count
);
```

Parameters

space
The color space to use for the gradient. You cannot use a pattern or indexed color space.

components
The color components for each color that defines the gradient. The components should be in the color space specified by `space`. If you are unsure of the number of components, you can call the function `CGColorSpaceGetNumberOfComponents` (page 54).

The number of items in this array should be the product of `count` and the number of components in the color space. For example, if the color space is an RGBA color space and you want to use two colors in the gradient (one for a starting location and another for an ending location), then you need to provide 8 values in `components`—red, green, blue, and alpha values for the first color, followed by red, green, blue, and alpha values for the second color.

locations
The location for each color provided in `components`. Each location must be a `CGFloat` value in the range of 0 to 1, inclusive. If 0 and 1 are not in the `locations` array, Quartz uses the colors provided that are closest to 0 and 1 for those locations.

If `locations` is `NULL`, the first color in `colors` is assigned to location `0`, the last color in`colors` is assigned to location `1`, and intervening colors are assigned locations that are at equal intervals in between.

count
The number of locations provided in the `locations` parameters.

Return Value
A CGGradient object.

Availability
Available in Mac OS X v10.5 and later.

See Also
`CGContextDrawLinearGradient` (page 86)
`CGContextDrawRadialGradient` (page 89)

Declared In
`CGGradient.h`

CGGradientCreateWithColors

Creates a CGGradient object from a color space and the provided color objects and locations.

```
CGGradientRef CGGradientCreateWithColors(
    CGColorSpaceRef space,
    CFArrayRef colors,
    const CGFloat locations[]
);
```

Parameters

space

The color space to use for the gradient. You cannot use a pattern or indexed color space.

colors

A non-empty array of CGColor objects that should be in the color space specified by `space`. If `space` is not `NULL`, each color will be converted (if necessary) to that color space and the gradient will drawn in that color space. Otherwise, each color will be converted to and drawn in the GenericRGB color space.

locations

The location for each color provided in `colors`; each location must be a `CGFloat` value in the range of 0 to 1, inclusive. If 0 and 1 are not in the `locations` array, Quartz uses the colors provided that are closest to 0 and 1 for those locations.

If `locations` is `NULL`, the first color in `colors` is assigned to location `0`, the last color in`colors` is assigned to location `1`, and intervening colors are assigned locations that are at equal intervals in between.

The `locations` array should contain the same number of items as the `colors` array.

Return Value
A CGGradient object.

Availability
Available in Mac OS X v10.5 and later.

See Also
`CGContextDrawLinearGradient` (page 86)
`CGContextDrawRadialGradient` (page 89)

Declared In
`CGGradient.h`

CGGradientGetTypeID

Returns the Core Foundation type identifier for CGGradient objects.

```
CFTypeID CGGradientGetTypeID (
   void
);
```

Return Value
The Core Foundation identifier for the opaque type `CGGradientRef`.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CGGradient.h`

CGGradientRelease

Decrements the retain count of a CGGradient object.

```
void CGGradientRelease (
   CGGradientRef gradient
);
```

Parameters
gradient
The gradient object to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the *gradient* parameter is `NULL`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGGradient.h`

CGGradientRetain

Increments the retain count of a CGGradient object.

```
CGGradientRef CGGradientRetain(
   CGGradientRef gradient
);
```

Parameters
gradient
The gradient object to retain.

Return Value
The same gradient object that you passed in as the `gradient` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `gradient` parameter is `NULL`.

Availability
Available in Mac OS X version 10.5 and later.

Declared In
`CGGradient.h`

Data Types

CGGradientRef

An opaque type that represents a Quartz gradient.

```
typedef struct CGGradient *CGGradientRef;
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGGradient.h`

Constants

Gradient Drawing Options

Drawing locations for gradients.

```
enum {
    kCGGradientDrawsBeforeStartLocation = (1 << 0),
    kCGGradientDrawsAfterEndLocation = (1 << 1)
};
typedef enum CGGradientDrawingOptions CGGradientDrawingOptions;
```

Constants

`kCGGradientDrawsBeforeStartLocation`
The fill should extend beyond the starting location. The color that extends beyond the starting point is the solid color defined by the CGGradient object to be at location 0.

Available in Mac OS X v10.5 and later.

Declared in `CGGradient.h`.

`kCGGradientDrawsAfterEndLocation`
The fill should extend beyond the ending location. The color that extends beyond the ending point is the solid color defined by the CGGradient object to be at location 1.

Available in Mac OS X v10.5 and later.

Declared in `CGGradient.h`.

Declared In
`CGGradient.h`

CGImage Reference

Derived From:	*CFType Reference*
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGImage.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGImageRef` opaque type represents bitmap images and bitmap image masks, based on sample data that you supply. A bitmap (or sampled) image is a rectangular array of pixels, with each pixel representing a single sample or data point in a source image.

Functions by Task

Creating Bitmap Images

`CGImageCreate` (page 211)
Creates a bitmap image from data supplied by a data provider.

`CGImageCreateCopy` (page 212)
Creates a copy of a bitmap image.

`CGImageCreateCopyWithColorSpace` (page 213)
Create a copy of a bitmap image, replacing its colorspace.

`CGImageCreateWithJPEGDataProvider` (page 214)
Creates a bitmap image using JPEG-encoded data supplied by a data provider.

`CGImageCreateWithPNGDataProvider` (page 216)
Creates a Quartz bitmap image using PNG-encoded data supplied by a data provider.

`CGImageCreateWithImageInRect` (page 213)
Creates a bitmap image using the data contained within a subregion of an existing bitmap image.

`CGImageCreateWithMask` (page 215)
Creates a bitmap image from an existing image and an image mask.

`CGImageCreateWithMaskingColors` (page 215)
Creates a bitmap image by masking an existing bitmap image with the provided color values.

Creating an Image Mask

`CGImageMaskCreate` (page 223)
Creates a bitmap image mask from data supplied by a data provider.

Retaining and Releasing Images

`CGImageRetain` (page 225)
Increments the retain count of a bitmap image.

`CGImageRelease` (page 224)
Decrements the retain count of a bitmap image.

Getting the CFType ID

`CGImageGetTypeID` (page 222)
Returns the type identifier for Quartz bitmap images.

Getting Information About an Image

`CGImageGetAlphaInfo` (page 217)
Returns the alpha channel information for a bitmap image.

`CGImageGetBitmapInfo` (page 217)
Returns the bitmap information for a bitmap image.

`CGImageGetBitsPerComponent` (page 218)
Returns the number of bits allocated for a single color component of a bitmap image.

`CGImageGetBitsPerPixel` (page 218)
Returns the number of bits allocated for a single pixel in a bitmap image.

`CGImageGetBytesPerRow` (page 219)
Returns the number of bytes allocated for a single row of a bitmap image.

`CGImageGetColorSpace` (page 219)
Return the color space for a bitmap image.

`CGImageGetDataProvider` (page 220)
Returns the data provider for a bitmap image.

`CGImageGetDecode` (page 220)
Returns the decode array for a bitmap image.

`CGImageGetHeight` (page 220)
Returns the height of a bitmap image.

`CGImageGetShouldInterpolate` (page 221)
Returns the interpolation setting for a bitmap image.

`CGImageGetRenderingIntent` (page 221)
Returns the rendering intent setting for a bitmap image.

`CGImageGetWidth` (page 222)
Returns the width of a bitmap image.

`CGImageIsMask` (page 223)
Returns whether a bitmap image is an image mask.

Functions

CGImageCreate

Creates a bitmap image from data supplied by a data provider.

```
CGImageRef CGImageCreate (
   size_t width,
   size_t height,
   size_t bitsPerComponent,
   size_t bitsPerPixel,
   size_t bytesPerRow,
   CGColorSpaceRef colorspace,
   CGBitmapInfo bitmapInfo,
   CGDataProviderRef provider,
   const CGFloat decode[],
   bool shouldInterpolate,
   CGColorRenderingIntent intent
);
```

Parameters

width
The width, in pixels, of the required image.

height
The height, in pixels, of the required image

bitsPerComponent
The number of bits for each component in a source pixel. For example, if the source image uses the RGBA-32 format, you would specify 8 bits per component.

bitsPerPixel
The total number of bits in a source pixel. This value must be at least `bitsPerComponent` times the number of components per pixel.

bytesPerRow
The number of bytes of memory for each horizontal row of the bitmap.

colorspace
The color space for the image. Quartz retains the color space you pass in; on return, you may safely release it.

bitmapInfo
A `CGBitmapInfo` constant that specifies whether the bitmap should contain an alpha channel and its relative location in a pixel, along with whether the components are floating-point or integer values.

provider
The source of data for the bitmap. For information about supported data formats, see the discussion below. Quartz retains this object; on return, you may safely release it.

decode

The decode array for the image. If you do not want to allow remapping of the image's color values, pass `NULL` for the decode array. For each color component in the image's color space, a decode array provides a pair of values denoting the upper and lower limits of a range. For example, the decode array for a source image in the RGB color space would contain six entries total, consisting of one pair each for red, green, and blue. When the image is rendered, Quartz uses a linear transform to map the original component value into a relative number within your designated range that is appropriate for the destination color space.

shouldInterpolate

A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply a pixel-smoothing algorithm to the image. Without interpolation, the image may appear jagged or pixelated when drawn on an output device with higher resolution than the image data.

intent

A rendering intent constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context. The rendering intent determines the exact method used to map colors from one color space to another. For descriptions of the defined rendering-intent constants, see `Color Rendering Intents` (page 59).

Return Value
A new Quartz bitmap image. You are responsible for releasing this object by calling `CGImageRelease` (page 224).

Discussion
The data provider should provide raw data that matches the format specified by the other input parameters. To use encoded data (for example, from a file specified by a URL-based data provider), see `CGImageCreateWithJPEGDataProvider` (page 214) and `CGImageCreateWithPNGDataProvider` (page 216). In Mac OS X version 10.3 and later, you can also use the QuickTime function `GraphicsImportCreateCGImage` to decode an image file in any supported format and create a CGImage, in a single operation.

For information on supported pixel formats, see *Quartz 2D Programming Guide*.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageCreateCopy

Creates a copy of a bitmap image.

```
CGImageRef CGImageCreateCopy (
   CGImageRef image
);
```

Parameters

image

The image to copy.

Return Value
An copy of the image specified by the `image` parameter.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGImage.h`

CGImageCreateCopyWithColorSpace

Create a copy of a bitmap image, replacing its colorspace.

```
CGImageRef CGImageCreateCopyWithColorSpace (
   CGImageRef image,
   CGColorSpaceRef colorspace
);
```

Parameters

image
The graphics image to copy.

colorspace
The destination color space. The number of components in this color space must be the same as the number in the specified image.

Return Value
A new Quartz image that is a copy of the image passed as the `image` parameter but with its color space replaced by that specified by the `colorspace` parameter. Returns `NULL` if `image` is an image mask, or if the number of components of `colorspace` is not the same as the number of components of the colorspace of `image`. You are responsible for releasing this object using `CGImageRelease` (page 224).

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGImage.h`

CGImageCreateWithImageInRect

Creates a bitmap image using the data contained within a subregion of an existing bitmap image.

```
CGImageRef CGImageCreateWithImageInRect (
   CGImageRef image,
   CGRect rect
);
```

Parameters

image
The image to extract the subimage from.

rect
A rectangle whose coordinates specify the area to create an image from.

Return Value
A CGImage object that specifies a subimage of the image. If the `rect` parameter defines an area that is not in the image, returns `NULL`.

Discussion
Quartz performs these tasks to create the subimage:

- Adjusts the area specified by the `rect` parameter to integral bounds by calling the function `CGRectIntegral`.
- Intersects the result with a rectangle whose origin is `(0,0)` and size is equal to the size of the image specified by the `image` parameter.
- References the pixels within the resulting rectangle, treating the first pixel within the rectangle as the origin of the subimage.

If `W` and `H` are the width and height of image, respectively, then the point `(0,0)` corresponds to the first pixel of the image data. The point `(W-1, 0)` is the last pixel of the first row of the image data while `(0, H-1)` is the first pixel of the last row of the image data and `(W-1, H-1)` is the last pixel of the last row of the image data.

The resulting image retains a reference to the original image, which means you may release the original image after calling this function.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGImage.h`

CGImageCreateWithJPEGDataProvider

Creates a bitmap image using JPEG-encoded data supplied by a data provider.

```
CGImageRef CGImageCreateWithJPEGDataProvider (
   CGDataProviderRef source,
   const CGFloat decode[],
   bool shouldInterpolate,
   CGColorRenderingIntent intent
);
```

Parameters

source
A data provider supplying JPEG-encoded data.

decode
The decode array for the image. Typically a decode array is unnecessary, and you should pass `NULL`.

shouldInterpolate
A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply a pixel-smoothing algorithm to the image.

intent
A `CGColorRenderingIntent` constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context.

Return Value
A new Quartz bitmap image. You are responsible for releasing this object by calling `CGImageRelease` (page 224).

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CGImage.h`

CGImageCreateWithMask

Creates a bitmap image from an existing image and an image mask.

```
CGImageRef CGImageCreateWithMask (
   CGImageRef image,
   CGImageRef mask
);
```

Parameters

image

The image to apply the `mask` parameter to. This image must not be an image mask and may not have an image mask or masking color associated with it.

mask

A mask. If the mask is an image, it must be in the DeviceGray color space, must not have an alpha component, and may not itself be masked by an image mask or a masking color. If the mask is not the same size as the image specified by the `image` parameter, then Quartz scales the mask to fit the image.

Return Value
An image created by masking `image` with `mask`. You are responsible for releasing this object by calling `CGImageRelease` (page 224).

Discussion
The resulting image depends on whether the `mask` parameter is an image mask or an image. If the `mask` parameter is an image mask, then the source samples of the image mask act as an inverse alpha value. That is, if the value of a source sample in the image mask is S, then the corresponding region in `image` is blended with the destination using an alpha value of (1-S). For example, if S is 1, then the region is not painted, while if S is 0, the region is fully painted.

If the `mask` parameter is an image, then it serves as an alpha mask for blending the image onto the destination. The source samples of `mask`' act as an alpha value. If the value of the source sample in mask is S, then the corresponding region in image is blended with the destination with an alpha of S. For example, if S is 0, then the region is not painted, while if S is 1, the region is fully painted.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGImage.h`

CGImageCreateWithMaskingColors

Creates a bitmap image by masking an existing bitmap image with the provided color values.

```
CGImageRef CGImageCreateWithMaskingColors (
   CGImageRef image,
   const CGFloat components[]
);
```

Parameters

image

The image to mask. This parameter may not be an image mask, may not already have an image mask or masking color associated with it, and cannot have an alpha component.

components

An array of color components that specify a color or range of colors to mask the image with. The array must contain 2N values { min[1], max[1], ... min[N], max[N] } where N is the number of components in color space of `image`. Each value in `components` must be a valid image sample value. If `image` has integer pixel components, then each value must be in the range [0 .. 2**`bitsPerComponent` - 1] (where `bitsPerComponent` is the number of bits/component of `image`). If `image` has floating-point pixel components, then each value may be any floating-point number which is a valid color component.

Return Value

An image created by masking `image` with the colors specified in the `components` array. You are responsible for releasing this object by calling `CGImageRelease` (page 224).

Discussion

Any image sample with color value {c[1], ... c[N]} where min[i] <= c[i] <= max[i] for 1 <= i <= N is masked out (that is, not painted). This means that anything underneath the unpainted samples, such as the current fill color, shows through.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGImage.h`

CGImageCreateWithPNGDataProvider

Creates a Quartz bitmap image using PNG-encoded data supplied by a data provider.

```
CGImageRef CGImageCreateWithPNGDataProvider (
   CGDataProviderRef source,
   const CGFloat decode[],
   bool shouldInterpolate,
   CGColorRenderingIntent intent
);
```

Parameters

source

A data provider supplying PNG-encoded data.

decode

The decode array for the image. Typically a decode array is unnecessary, and you should pass `NULL`.

shouldInterpolate

A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply a pixel-smoothing algorithm to the image.

intent

A `CGColorRenderingIntent` constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context.

Return Value
A new Quartz bitmap image. You are responsible for releasing this object by calling `CGImageRelease` (page 224).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGImage.h`

CGImageGetAlphaInfo

Returns the alpha channel information for a bitmap image.

```
CGImageAlphaInfo CGImageGetAlphaInfo (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value
A `CGImageAlphaInfo` constant that specifies (1) whether the bitmap contains an alpha channel, (2) where the alpha bits are located in the image data, and (3) whether the alpha value is premultiplied. For possible values, see "Constants" (page 226). The function returns `kCGImageAlphaNone` if the `image` parameter refers to an image mask.

Discussion
The alpha value is what determines the opacity of a pixel when it is drawn.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageGetBitmapInfo

Returns the bitmap information for a bitmap image.

```
CGBitmapInfo CGImageGetBitmapInfo (
   CGImageRef image
);
```

Parameters

image

An image.

Return Value
The bitmap information associated with an image.

Discussion
This function returns a constant that specifies:

- The type of bitmap data—floating point or integer. You use the constant `kCGBitmapFloatComponents` to extract this information.
- Whether an alpha channel is in the data, and if so, how the alpha data is stored. You use the constant `kCGBitmapAlphaInfoMask` to extract the alpha information. Alpha information is specified as one of the constants listed in "Alpha Information for Images" (page 226).

You can extract the alpha information

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGImage.h`

CGImageGetBitsPerComponent

Returns the number of bits allocated for a single color component of a bitmap image.

```
size_t CGImageGetBitsPerComponent (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
The number of bits used in memory for each color component of the specified bitmap image (or image mask). Possible values are 1, 2, 4, or 8. For example, for a 16-bit RGB(A) colorspace, the function would return a value of 4 bits per color component.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageGetBitsPerPixel

Returns the number of bits allocated for a single pixel in a bitmap image.

```
size_t CGImageGetBitsPerPixel (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
The number of bits used in memory for each pixel of the specified bitmap image (or image mask).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageGetBytesPerRow

Returns the number of bytes allocated for a single row of a bitmap image.

```
size_t CGImageGetBytesPerRow (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
The number of bytes used in memory for each row of the specified bitmap image (or image mask).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageGetColorSpace

Return the color space for a bitmap image.

```
CGColorSpaceRef CGImageGetColorSpace (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
The source color space for the specified bitmap image, or `NULL` if the image is an image mask. You are responsible for retaining and releasing the color space as necessary.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageGetDataProvider

Returns the data provider for a bitmap image.

```
CGDataProviderRef CGImageGetDataProvider (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

The data provider for the specified bitmap image (or image mask). You are responsible for retaining and releasing the data provider as necessary.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGImage.h`

CGImageGetDecode

Returns the decode array for a bitmap image.

```
const CGFloat * CGImageGetDecode (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

The decode array for a bitmap image (or image mask). See the discussion for a description of possible return values.

Discussion

For a bitmap image or image mask, for each color component in the source color space, the decode array contains a pair of values denoting the upper and lower limits of a range. When the image is rendered, Quartz uses a linear transform to map the original component value into a relative number, within the designated range, that is appropriate for the destination color space. If remapping of the image's color values is not allowed, the returned value will be `NULL`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGImage.h`

CGImageGetHeight

Returns the height of a bitmap image.

```
size_t CGImageGetHeight (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value
The height in pixels of the bitmap image (or image mask).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`CGImage.h`

CGImageGetRenderingIntent

Returns the rendering intent setting for a bitmap image.

```
CGColorRenderingIntent CGImageGetRenderingIntent (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value
Returns the `CGColorRenderingIntent` constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context in which the image is drawn. If the image is an image mask, this function returns `kCGRenderingIntentDefault`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageGetShouldInterpolate

Returns the interpolation setting for a bitmap image.

```
bool CGImageGetShouldInterpolate (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value
Returns `1` if interpolation is enabled for the specified bitmap image (or image mask), otherwise, returns `0`.

Discussion
The interpolation setting specifies whether Quartz should apply an edge-smoothing algorithm to the associated image.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageGetTypeID

Returns the type identifier for Quartz bitmap images.

```
CFTypeID CGImageGetTypeID (
   void
);
```

Return Value
The identifier for the opaque type `CGImageRef` (page 225).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGImage.h`

CGImageGetWidth

Returns the width of a bitmap image.

```
size_t CGImageGetWidth (
   CGImageRef image
);
```

Parameters
image
The image to examine.

Return Value
The width, in pixels, of the specified bitmap image (or image mask).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`CGImage.h`

CGImageIsMask

Returns whether a bitmap image is an image mask.

```
bool CGImageIsMask (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

A Boolean value that indicates whether the image passed in the `image` parameter is an image mask (`true` indicates that the image is an image mask).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGImage.h`

CGImageMaskCreate

Creates a bitmap image mask from data supplied by a data provider.

```
CGImageRef CGImageMaskCreate (
   size_t width,
   size_t height,
   size_t bitsPerComponent,
   size_t bitsPerPixel,
   size_t bytesPerRow,
   CGDataProviderRef provider,
   const CGFloat decode[],
   bool shouldInterpolate
);
```

Parameters

width

The width, in pixels, of the required image mask.

height

The height, in pixels, of the required image mask.

bitsPerComponent

The number of significant masking bits in a source pixel. For example, if the source image is an 8-bit mask, you specify 8 bits per component. Image masks must be 1, 2, 4, or 8 bits per component.

bitsPerPixel

The total number of bits in a source pixel.

bytesPerRow

The number of bytes to use for each horizontal row of the image mask.

provider

The data source for the image mask.

decode

Typically a decode array is unnecessary, and you should pass `NULL`.

shouldInterpolate

A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply an edge-smoothing algorithm to the image mask.

Return Value
A Quartz bitmap image mask. You are responsible for releasing this object by calling `CGImageRelease` (page 224).

Discussion
A Quartz bitmap image mask is used the same way an artist uses a silkscreen, or a sign painter uses a stencil. The bitmap represents a mask through which a color is transferred. The bitmap itself does not have a color. It gets its color from the fill color currently set in the graphics state.

When you draw into a context with a bitmap image mask, Quartz uses the mask to determine where and how the current fill color is applied to the image rectangle. Each sample value in the mask specifies how much of the current fill color is masked out at a specific location. Effectively, the sample value specifies the opacity of the mask. Larger values represent greater opacity and hence less color applied to the page.

Image masks must be 1, 2, 4, or 8 bits per component. For a 1-bit mask, a sample value of `1` specifies sections of the mask that are masked out; these sections block the current fill color. A sample value of `0` specifies sections of the mask that are not masked out; these sections show the current fill color of the graphics state when the mask is painted. You can think of the sample values as an inverse alpha. That is, a value of `1` is transparent and `0` is opaque.

For image masks that are 2, 4, or 8 bits per component, each component is mapped to a range of 0 to 1 by scaling using this formula:

```
1/(2^bits per component - 1)
```

For example, a 4-bit mask has values that range from 0 to 15. These values are scaled by 1/15 so that each component ranges from 0 to 1. Component values that rescale to 0 or 1 behave the same way as they behave for 1-bit image masks. Values that scale to between 0 and 1 act as an inverse alpha. That is, the fill color is painted as if it has an alpha value of (`1 - MaskSampleValue`). For example, if the sample value of an 8-bit mask scales to `0.8`, the current fill color is painted as if it has an alpha value of `0.2`, that is (`1-0.8`).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

CGImageRelease

Decrements the retain count of a bitmap image.

```
void CGImageRelease (
   CGImageRef image
);
```

Parameters

image

The image to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `image` parameter is `NULL`.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
WhackedTV

Declared In
`CGImage.h`

CGImageRetain

Increments the retain count of a bitmap image.

```
CGImageRef CGImageRetain (
   CGImageRef image
);
```

Parameters

image
The image to retain.

Return Value
The same image you passed in as the `image` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `image` parameter is `NULL`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGImage.h`

Data Types

CGImageRef

An opaque type that encapsulates bitmap image information.

```
typedef struct CGImage *CGImageRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGImage.h`

Constants

Alpha Information for Images

Storage options for alpha component data.

```
enum CGImageAlphaInfo {
    kCGImageAlphaNone,
    kCGImageAlphaPremultipliedLast,
    kCGImageAlphaPremultipliedFirst,
    kCGImageAlphaLast,
    kCGImageAlphaFirst,
    kCGImageAlphaNoneSkipLast,
    kCGImageAlphaNoneSkipFirst
};
typedef enum CGImageAlphaInfo CGImageAlphaInfo;
```

Constants

`kCGImageAlphaFirst`

The alpha component is stored in the most significant bits of each pixel. For example, non-premultiplied ARGB.

Available in Mac OS X v10.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaLast`

The alpha component is stored in the least significant bits of each pixel. For example, non-premultiplied RGBA.

Available in Mac OS X v10.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaNone`

There is no alpha channel. If the total size of the pixel is greater than the space required for the number of color components in the color space, the least significant bits are ignored. This value is equivalent to `kCGImageAlphaNoneSkipLast`.

Available in Mac OS X v10.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaNoneSkipFirst`

There is no alpha channel. If the total size of the pixel is greater than the space required for the number of color components in the color space, the most significant bits are ignored.

Available in Mac OS X v10.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaOnly`

There is no color data, only an alpha channel.

Available in Mac OS X v10.3 and later.

Declared in `CGImage.h`.

`kCGImageAlphaNoneSkipLast`
: There is no alpha channel. If the total size of the pixel is greater than the space required for the number of color components in the color space, the least significant bits are ignored. This value is equivalent to `kCGImageAlphaNone`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGImage.h`.

`kCGImageAlphaPremultipliedFirst`
: The alpha component is stored in the most significant bits of each pixel and the color components have already been multiplied by this alpha value. For example, premultiplied ARGB.

  Available in Mac OS X v10.0 and later.

  Declared in `CGImage.h`.

`kCGImageAlphaPremultipliedLast`
: The alpha component is stored in the least significant bits of each pixel and the color components have already been multiplied by this alpha value. For example, premultiplied RGBA.

  Available in Mac OS X v10.0 and later.

  Declared in `CGImage.h`.

Discussion
A `CGImageAlphaInfo` constant specifies (1) whether a bitmap contains an alpha channel, (2) where the alpha bits are located in the image data, and (3) whether the alpha value is premultiplied. You can obtain a `CGImageAlphaInfo` constant for an image by calling the function `CGImageGetAlphaInfo` (page 217). (You provide a `CGBitmapInfo` constant to the function `CGImageCreate` (page 211), part of which is a `CGImageAlphaInfo` constant.)

Quartz accomplishes alpha blending by combining the color components of the source image with the color components of the destination image using the linear interpolation formula, where "source" is one color component of one pixel of the new paint and "destination" is one color component of the background image.

Quartz supports premultiplied alpha only for images. You should not premultiply any other color values specified in Quartz.

Declared In
`CGImage.h`

Image Bitmap Information

Component information for a bitmap image.

```
enum {
    kCGBitmapAlphaInfoMask = 0x1F,
    kCGBitmapFloatComponents = (1 << 8),

    kCGBitmapByteOrderMask = 0x7000,
    kCGBitmapByteOrderDefault = (0 << 12),
    kCGBitmapByteOrder16Little = (1 << 12),
    kCGBitmapByteOrder32Little = (2 << 12),
    kCGBitmapByteOrder16Big = (3 << 12),
    kCGBitmapByteOrder32Big = (4 << 12)
};
typedef uint32_t CGBitmapInfo;
```

```
#ifdef __BIG_ENDIAN__
    kCGBitmapByteOrder16Host kCGBitmapByteOrder16Big
    kCGBitmapByteOrder32Host kCGBitmapByteOrder32Big
#else
    kCGBitmapByteOrder16Host kCGBitmapByteOrder16Little
    kCGBitmapByteOrder32Host kCGBitmapByteOrder32Little
#endif
```

Constants

`kCGBitmapAlphaInfoMask`
The alpha information mask. Use this to extract alpha information that specifies whether a bitmap contains an alpha channel and how the alpha channel is generated.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapFloatComponents`
The components of a bitmap are floating-point values.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrderMask`
The byte ordering of pixel formats.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrderDefault`
The default byte order.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder16Little`
16-bit, little endian format.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder32Little`
32-bit, little endian format.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder16Big`
16-bit, big endian format.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder32Big`
32-bit, big endian format.

Available in Mac OS X v10.4 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder16Host`
16-bit, host endian format.

`kCGBitmapByteOrder32Host`
32-bit, host endian format.

Discussion
Applications that store pixel data in memory using ARGB format must take care in how they read data. If the code is not written correctly, it's possible to misread the data which leads to colors or alpha that appear wrong. The Quartz byte order constants specify the byte ordering of pixel formats. To specify byte ordering to Quartz use a bitwise OR operator to combine the appropriate constant with the `bitmapInfo` parameter.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGImage.h`

CGImageDestination Reference

Derived From:	CFType
Framework:	ApplicationServices/ImageIO
Declared in	CGImageDestination.h
Companion guide	Quartz 2D Programming Guide

Overview

CGImageDestination objects, available in Mac OS X v10.4 or later, abstract the data-writing task. An image destination can represent a single image or multiple images. It can contain thumbnail images as well as properties for each image.

The functions described in this reference can write data to three kinds of destinations: a URL, a `CFData` object, and a data consumer. After creating a CGImageDestination object for the appropriate destination, you can add image data and set image properties. When you are finished adding data, call the function `CGImageDestinationFinalize` to write the image data and properties to the URL, `CFData` object, or data consumer.

Functions by Task

Creating Image Destinations

`CGImageDestinationCreateWithDataConsumer` (page 234)
Creates an image destination that writes to the specified data consumer.

`CGImageDestinationCreateWithData` (page 234)
Creates an image destination that writes to a Core Foundation mutable data object.

`CGImageDestinationCreateWithURL` (page 235)
Creates an image destination that writes to a location specified by a URL.

Adding Images

`CGImageDestinationAddImage` (page 232)
Adds an image to an image destination.

`CGImageDestinationAddImageFromSource` (page 233)
Adds an image from an image source to an image destination.

Getting Type Identifiers

`CGImageDestinationCopyTypeIdentifiers` (page 233)
Returns an array of the uniform type identifiers (UTIs) that are supported for image destinations.

`CGImageDestinationGetTypeID` (page 236)
Returns the unique type identifier of an image destination opaque type.

Setting Properties

`CGImageDestinationSetProperties` (page 236)
Applies one or more properties to all images in an image destination.

Finalizing an Image Destination

`CGImageDestinationFinalize` (page 235)
Writes image data and properties to the data, URL, or data consumer associated with the image destination.

Functions

CGImageDestinationAddImage

Adds an image to an image destination.

```
void CGImageDestinationAddImage (
    CGImageDestinationRef idst,
    CGImageRef image,
    CFDictionaryRef properties
);
```

Parameters

idst
An image destination

image
The image to add.

properties
An optional dictionary that specifies the properties of the added image. The dictionary can contain any of the properties described in "Destination Properties" (page 237) or the image properties described in *CGImageProperties Reference*.

Discussion
The function logs an error if you add more images than what you specified when you created the image destination.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageDestination.h

CGImageDestinationAddImageFromSource

Adds an image from an image source to an image destination.

```
void CGImageDestinationAddImageFromSource (
    CGImageDestinationRef idst,
    CGImageSourceRef isrc,
    size_t index,
    CFDictionaryRef properties
);
```

Parameters

idst
An image destination.

isrc
An image source.

index
An index that specifies the location of the image in the image source. The index is zero-based.

properties
A dictionary that specifies properties to overwrite or add to the source image properties. If a key in `properties` has the value `kCFNull`, the corresponding property in the image destination is removed. The dictionary can contain any of the properties described in "Destination Properties" (page 237) or the image properties described in *CGImageProperties Reference*.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageDestination.h

CGImageDestinationCopyTypeIdentifiers

Returns an array of the uniform type identifiers (UTIs) that are supported for image destinations.

```
CFArrayRef CGImageDestinationCopyTypeIdentifiers (
    void
);
```

Return Value
Returns an array of the UTIs that are supported for image destinations. See Uniform Type Identifiers Overview for a list of system-declared and third-party UTIs that can be returned.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageDestination.h

CGImageDestinationCreateWithData

Creates an image destination that writes to a Core Foundation mutable data object.

```
CGImageDestinationRef CGImageDestinationCreateWithData (
    CFMutableDataRef data,
    CFStringRef type,
    size_t count,
    CFDictionaryRef options
);
```

Parameters

data

The data object to write to. For more information on data objects, see *CFData Reference* and Data Objects.

type

The uniform type identifier (UTI) of the resulting image file. See *Uniform Type Identifiers Overview* for a list of system-declared and third-party UTIs.

count

The number of images (not including thumbnail images) that the image file will contain.

options

Reserved for future use. Pass `NULL`.

Return Value

An image destination. You are responsible for releasing this object using `CFRelease`.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGImageDestination.h`

CGImageDestinationCreateWithDataConsumer

Creates an image destination that writes to the specified data consumer.

```
CGImageDestinationRef CGImageDestinationCreateWithDataConsumer (
    CGDataConsumerRef consumer,
    CFStringRef type,
    size_t count,
    CFDictionaryRef options
);
```

Parameters

consumer

The data consumer to write to. For information on data consumers see *CGDataConsumer Reference* and *Quartz 2D Programming Guide*.

type

The uniform type identifier (UTI) of the resulting image file. See *Uniform Type Identifiers Overview* for a list of system-declared and third-party UTIs.

count

The number of images (not including thumbnail images) that the image file will contain.

options
Reserved for future use. Pass `NULL`.

Return Value
An image destination. You are responsible for releasing this object using `CFRelease`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageDestination.h`

CGImageDestinationCreateWithURL

Creates an image destination that writes to a location specified by a URL.

```
CGImageDestinationRef CGImageDestinationCreateWithURL (
    CFURLRef url,
    CFStringRef type,
    size_t count,
    CFDictionaryRef options
);
```

Parameters

url
The URL to write to. If the URL already exists, the data at this location is overwritten.

type
The UTI (uniform type identifier) of the resulting image file. See *Uniform Type Identifiers Overview* for a list of system-declared and third-party UTIs.

count
The number of images (not including thumbnail images) that the image file will contain.

options
Reserved for future use. Pass `NULL`.

Return Value
An image destination. You are responsible for releasing this object using `CFRelease`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageDestination.h`

CGImageDestinationFinalize

Writes image data and properties to the data, URL, or data consumer associated with the image destination.

```
bool CGImageDestinationFinalize (
   CGImageDestinationRef idst
);
```

Parameters

idst

An image destination.

Return Value
Returns `true` if the image is successfully written; `false` otherwise.

Discussion
You must call this function or the output of the image destination will not be valid. After calling this function, no additional data can be added to the image destination.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageDestination.h`

CGImageDestinationGetTypeID

Returns the unique type identifier of an image destination opaque type.

```
CFTypeID CGImageDestinationGetTypeID (
    void
);
```

Return Value
Returns the Core Foundation type ID for an image destination.

Discussion
A type identifier is an integer that identifies the opaque type to which a Core Foundation object belongs. You use type IDs in various contexts, such as when you are operating on heterogeneous collections.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageDestination.h`

CGImageDestinationSetProperties

Applies one or more properties to all images in an image destination.

```
void CGImageDestinationSetProperties (
    CGImageDestinationRef idst,
    CFDictionaryRef properties
);
```

Parameters

idst

An image destination.

properties
A dictionary that contains the properties to apply. You can set any of the properties described in "Destination Properties" (page 237) or the image properties described in *CGImageProperties Reference*.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageDestination.h

Data Types

CGImageDestinationRef

An opaque type that represents an image destination.

```
typedef struct CGImageDestination *CGImageDestinationRef;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGImageDestination.h

Constants

Destination Properties

Properties for a single image in an image destination.

```
const CFStringRef kCGImageDestinationLossyCompressionQuality
const CFStringRef kCGImageDestinationBackgroundColor
```

Constants
kCGImageDestinationLossyCompressionQuality
The desired compression quality to use when writing to an image destination. If present, the value associated with this key must be a `CFNumberRef` data type in the range `0.0` to `1.0`. A value of `1.0` specifies to use lossless compression if destination format supports it. A value of 0.0 implies to use maximum compression.

Available in Mac OS X v10.4 and later.

Declared in `CGImageDestination.h`.

`kCGImageDestinationBackgroundColor`

The desired background color to composite against when writing an image that has an alpha component to a destination format that does not support alpha. If present, the value associated with this key must be a `CGColorRef` (page 41) data type without an alpha component of its own. If not present, and if a background color is needed, a white color is used.

Available in Mac OS X v10.4 and later.

Declared in `CGImageDestination.h`.

Declared In
`CGImageDestination.h`

CGImageSource Reference

Derived From:	CFType
Framework:	ApplicationServices/ImageIO
Declared in	CGImageSource.h
Companion guides	Quartz 2D Programming Guide CGImage Reference

Overview

CGImageSource objects, available in Mac OS X v10.4 or later, abstract the data-reading task. An image source can read image data from a URL, a `CFData` object, or a data consumer.

After creating a CGImageSource object for the appropriate source, you can obtain images, thumbnails, image properties, and other image information using CGImageSource functions.

Functions by Task

Creating an Image Source

`CGImageSourceCreateWithDataProvider` (page 244)
Creates an image source that reads data from the specified data provider.

`CGImageSourceCreateWithData` (page 244)
Creates an image source that reads from a Core Foundation data object.

`CGImageSourceCreateWithURL` (page 245)
Creates an image source that reads from a location specified by a URL.

Creating Images From an Image Source

`CGImageSourceCreateImageAtIndex` (page 242)
Creates a CGImage object for the image data associated with the specified index in an image source.

`CGImageSourceCreateThumbnailAtIndex` (page 243)
Creates a thumbnail image of the image located at a specified location in an image source.

`CGImageSourceCreateIncremental` (page 242)
Create an incremental image source.

Updating an Image Source

`CGImageSourceUpdateData` (page 248)
Updates an incremental image source with new data.

`CGImageSourceUpdateDataProvider` (page 248)
Updates an incremental image source with a new data provider.

Getting Information From an Image Source

`CGImageSourceGetTypeID` (page 247)
Returns the unique type identifier of an image source opaque type.

`CGImageSourceGetType` (page 247)
Returns the uniform type identifier of the source container.

`CGImageSourceCopyTypeIdentifiers` (page 241)
Returns an array of uniform type identifiers (UTIs) that are supported for image sources.

`CGImageSourceGetCount` (page 245)
Returns the number of images (not including thumbnails) in the image source.

`CGImageSourceCopyProperties` (page 240)
Returns the properties of the image source.

`CGImageSourceCopyPropertiesAtIndex` (page 241)
Returns the properties of the image at a specified location in an image source.

`CGImageSourceGetStatus` (page 246)
Return the status of an image source.

`CGImageSourceGetStatusAtIndex` (page 246)
Returns the current status of an image that is at a specified location in an image source.

Functions

CGImageSourceCopyProperties

Returns the properties of the image source.

```
CFDictionaryRef CGImageSourceCopyProperties (
   CGImageSourceRef isrc,
   CFDictionaryRef options
);
```

Parameters

isrc
An image source.

options
A dictionary you can use to request additional options. See "Image Source Option Dictionary Keys" (page 250) for the keys you can supply.

Return Value
A dictionary that contains the properties associated with the image source container. See *CGImageProperties Reference* for a list of properties that can be in the dictionary.

Discussion
These properties apply to the container in general but not necessarily to any individual image contained in the image source.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageSource.h`

CGImageSourceCopyPropertiesAtIndex

Returns the properties of the image at a specified location in an image source.

```
CFDictionaryRef CGImageSourceCopyPropertiesAtIndex (
   CGImageSourceRef isrc,
   size_t index,
   CFDictionaryRef options
);
```

Parameters

isrc
An image source.

index
The index of the image whose properties you want to obtain. The index is zero-based.

options
A dictionary you can use to request additional options. See "Image Source Option Dictionary Keys" (page 250) for the keys you can supply.

Return Value
A dictionary that contains the properties associated with the image. See *CGImageProperties Reference* for a list of properties that can be in the dictionary.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageSource.h`

CGImageSourceCopyTypeIdentifiers

Returns an array of uniform type identifiers (UTIs) that are supported for image sources.

```
CFArrayRef CGImageSourceCopyTypeIdentifiers (
   void
);
```

Return Value
Returns an array of the UTIs that are supported for image sources.

Discussion
See Uniform Type Identifiers Overview for a list of system-declared and third-party UTIs.

Availability
Available in Mac OS X version 10.4 and later.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`CGImageSource.h`

CGImageSourceCreateImageAtIndex

Creates a CGImage object for the image data associated with the specified index in an image source.

```
CGImageRef CGImageSourceCreateImageAtIndex (
   CGImageSourceRef isrc,
   size_t index,
   CFDictionaryRef options
);
```

Parameters

isrc

An image source.

index

The index that specifies the location of the image. The index is zero-based.

options

A dictionary that specifies additional creation options. See "`Image Source Option Dictionary Keys`" (page 250) for the keys you can supply.

Return Value
Returns a CGImage object. You are responsible for releasing this object using `CGImageRelease` (page 224).

Availability
Available in Mac OS X version 10.4 and later.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`CGImageSource.h`

CGImageSourceCreateIncremental

Create an incremental image source.

```
CGImageSourceRef CGImageSourceCreateIncremental (
   CFDictionaryRef options
);
```

Parameters

options

A dictionary that specifies additional creation options. See "Image Source Option Dictionary Keys" (page 250) for the keys you can supply.

Return Value

Returns an image source object. You are responsible for releasing this object using `CFRelease`.

Discussion

The function `CGImageSourceCreateIncremental` creates an empty image source container to which you can add data later by calling the functions `CGImageSourceUpdateDataProvider` or `CGImageSourceUpdateData`. You don't provide data when you call this function.

An incremental image is an image that is created in chunks, similar to the way large images viewed over the web are loaded piece by piece.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGImageSource.h`

CGImageSourceCreateThumbnailAtIndex

Creates a thumbnail image of the image located at a specified location in an image source.

```
CGImageRef CGImageSourceCreateThumbnailAtIndex (
   CGImageSourceRef isrc,
   size_t index,
   CFDictionaryRef options
);
```

Parameters

isrc

An image source.

index

The index that specifies the location of the image. The index is zero-based.

options

A dictionary that specifies additional creation options. See "Image Source Option Dictionary Keys" (page 250) for the keys you can supply.

Return Value

A CGImage object. You are responsible for releasing this object using `CGImageRelease` (page 224).

Discussion

If the image source is a PDF, this function creates a 72 dpi image of the PDF page specified by the index that you pass. You must, however, pass an options dictionary that contains either the `kCGImageSourceCreateThumbnailFromImageIfAbsent` or `kCGImageSourceCreateThumbnailFromImageAlways` keys, with the value of the key set to `TRUE`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageSource.h`

CGImageSourceCreateWithData

Creates an image source that reads from a Core Foundation data object.

```
CGImageSourceRef CGImageSourceCreateWithData (
   CFDataRef data,
   CFDictionaryRef options
);
```

Parameters

data

The data object to read from. For more information on data objects, see *CFData Reference* and Data Objects.

options

A dictionary that specifies additional creation options. See "`Image Source Option Dictionary Keys`" (page 250) for the keys you can supply.

Return Value
An image source. You are responsible for releasing this object using `CFRelease`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageSource.h`

CGImageSourceCreateWithDataProvider

Creates an image source that reads data from the specified data provider.

```
CGImageSourceRef CGImageSourceCreateWithDataProvider (
   CGDataProviderRef provider,
   CFDictionaryRef options
);
```

Parameters

provider

The data provider to read from. For more information on data providers, see *CGDataProvider Reference* and *Quartz 2D Programming Guide*.

options

A dictionary that specifies additional creation options. See "`Image Source Option Dictionary Keys`" (page 250) for the keys you can supply.

Return Value
An image source. You are responsible for releasing this object using `CFRelease`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGImageSource.h`

CGImageSourceCreateWithURL

Creates an image source that reads from a location specified by a URL.

```
CGImageSourceRef CGImageSourceCreateWithURL (
   CFURLRef url,
   CFDictionaryRef options
);
```

Parameters

url

The URL to read from.

options

A dictionary that specifies additional creation options. See "Image Source Option Dictionary Keys" (page 250) for the keys you can supply.

Return Value
An image source. You are responsible for releasing this object using `CFRelease`.

Availability
Available in Mac OS X version 10.4 and later.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`CGImageSource.h`

CGImageSourceGetCount

Returns the number of images (not including thumbnails) in the image source.

```
size_t CGImageSourceGetCount (
   CGImageSourceRef isrc
);
```

Parameters

isrc

An image source.

Return Value
The number of images. If the image source is a multilayered PSD file, the function returns `1`.

Discussion
This function does not extract the layers of a PSD file.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageSource.h

CGImageSourceGetStatus

Return the status of an image source.

```
CGImageSourceStatus CGImageSourceGetStatus (
   CGImageSourceRef isrc
);
```

Parameters

isrc
An image source.

Return Value
Returns the current status of the image source. See "Image Source Status" (page 249) for a list of possible values.

Discussion
The status is particularly informative for incremental image sources, but may also be used by clients that provide non-incremental data.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageSource.h

CGImageSourceGetStatusAtIndex

Returns the current status of an image that is at a specified location in an image source.

```
CGImageSourceStatus CGImageSourceGetStatusAtIndex (
   CGImageSourceRef isrc,
   size_t index
);
```

Parameters

isrc
An image source.

index
The index of the image whose status you want to obtain. The index is zero-based.

Return Value
Returns the current status of the image. See "Image Source Status" (page 249) for a list of possible values.

Discussion
The status is particularly informative for incremental image sources, but may also be used by clients that provide non-incremental data.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageSource.h

CGImageSourceGetType

Returns the uniform type identifier of the source container.

```
CFStringRef CGImageSourceGetType (
   CGImageSourceRef isrc
);
```

Parameters

isrc
An image source.

Return Value
The uniform type identifier of the image.

Discussion
The uniform type identifier (UTI) of the source container can be different from the type of the images in the container. For example, the `.icns` format supports embedded `JPEG2000`. The type of the source container is `"com.apple.icns"` but type of the images is `JPEG2000`.

See Uniform Type Identifier Concepts for a list of system-declared and third-party UTIs.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageSource.h

CGImageSourceGetTypeID

Returns the unique type identifier of an image source opaque type.

```
CFTypeID CGImageSourceGetTypeID (
   void
);
```

Return Value
Returns the Core Foundation type ID for an image source.

Discussion
A type identifier is an integer that identifies the opaque type to which a Core Foundation object belongs. You use type IDs in various contexts, such as when you are operating on heterogeneous collections. Note that a CFType ID is different from a uniform type identifier (UTI).

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGImageSource.h

CGImageSourceUpdateData

Updates an incremental image source with new data.

```
void CGImageSourceUpdateData (
   CGImageSourceRef isrc,
   CFDataRef data,
   bool final
);
```

Parameters

isrc

An image source.

data

The data to add to the image source. Each time you call the function `CGImageSourceUpdateData`, the `data` parameter must contain all of the image file data accumulated so far.

final

A value that specifies whether the data is the final set. Pass `true` if it is, `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGImageSource.h`

CGImageSourceUpdateDataProvider

Updates an incremental image source with a new data provider.

```
void CGImageSourceUpdateDataProvider (
   CGImageSourceRef isrc,
   CGDataProviderRef provider,
   bool final
);
```

Parameters

isrc

An image source.

provider

The new data provider. The new data provider must provide all the previous data supplied to the image source plus any additional new data.

final

A value that specifies whether the data is the final set. Pass `true` if it is, `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGImageSource.h`

Data Types

CGImageSourceRef

An opaque type that represents an image source.

```
typedef struct CGImageSource *CGImageSourceRef;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGImageSource.h`

Constants

Image Source Status

Status states for images and image sources.

```
enum CGImageSourceStatus {
    kCGImageStatusUnexpectedEOF = -5,
    kCGImageStatusInvalidData = -4,
    kCGImageStatusUnknownType = -3,
    kCGImageStatusReadingHeader = -2,
    kCGImageStatusIncomplete = -1,
    kCGImageStatusComplete = 0
};
typedef enum CGImageSourceStatus CGImageSourceStatus;
```

Constants

`kCGImageStatusUnexpectedEOF`
The end of the file was encountered unexpectedly.
Available in Mac OS X v10.4 and later.
Declared in `CGImageSource.h`.

`kCGImageStatusInvalidData`
The data is not valid.
Available in Mac OS X v10.4 and later.
Declared in `CGImageSource.h`.

`kCGImageStatusUnknownType`
The image is an unknown type.
Available in Mac OS X v10.4 and later.
Declared in `CGImageSource.h`.

`kCGImageStatusReadingHeader`
In the process of reading the header.

Available in Mac OS X v10.4 and later.

Declared in `CGImageSource.h`.

`kCGImageStatusIncomplete`
The operation is not complete

Available in Mac OS X v10.4 and later.

Declared in `CGImageSource.h`.

`kCGImageStatusComplete`
The operation is complete.

Available in Mac OS X v10.4 and later.

Declared in `CGImageSource.h`.

Discussion
These status values are returned by the functions `CGImageSourceGetStatus` (page 246) and `CGImageSourceGetStatusAtIndex` (page 246).

Declared In
`CGImageSource.h`

Image Source Option Dictionary Keys

Keys that you can include in the options dictionary to create an image source.

```
CFStringRef kCGImageSourceTypeIdentifierHint;
CFStringRef kCGImageSourceShouldAllowFloat;
CFStringRef kCGImageSourceShouldCache;
CFStringRef kCGImageSourceCreateThumbnailFromImageIfAbsent;
CFStringRef kCGImageSourceCreateThumbnailFromImageAlways;
CFStringRef kCGImageSourceThumbnailMaxPixelSize;
CFStringRef kCGImageSourceCreateThumbnailWithTransform
```

Constants
`kCGImageSourceTypeIdentifierHint`
The best guess of the uniform type identifier (UTI) for the format of the image source file. If specified, the value of this key must be a CFString object. This key can be provided in the options dictionary when you create a CGImageSource object.

Available in Mac OS X v10.4 and later.

Declared in `CGImageSource.h`.

`kCGImageSourceShouldAllowFloat`
Whether the image should be returned as a CGImage object that uses floating-point values, if supported by the file format. CGImage objects that use extended-range floating-point values may require additional processing to render in a pleasing manner. The value of this key must be a CFBoolean value. The default value is `kCFBooleanFalse`.

Available in Mac OS X v10.4 and later.

Declared in `CGImageSource.h`.

`kCGImageSourceShouldCache`
: Whether the image should be cached in a decoded form. The value of this key must be a CFBoolean value. The default value is `kCFBooleanTrue`. This key can be provided in the options dictionary that you can pass to the functions `CGImageSourceCopyPropertiesAtIndex` (page 241) and `CGImageSourceCreateImageAtIndex` (page 242).

  Available in Mac OS X v10.4 and later.

  Declared in `CGImageSource.h`.

`kCGImageSourceCreateThumbnailFromImageIfAbsent`
: Whether a thumbnail should be automatically created for an image if a thumbnail isn't present in the image source file. The thumbnail is created from the full image, subject to the limit specified by `kCGImageSourceThumbnailMaxPixelSize`. If a maximum pixel size isn't specified, then the thumbnail is the size of the full image, which in most cases is not desirable. This key must be a CFBoolean value. The default value is `kCFBooleanFalse`. This key can be provided in the options dictionary that you pass to the function `CGImageSourceCreateThumbnailAtIndex` (page 243).

  Available in Mac OS X v10.4 and later.

  Declared in `CGImageSource.h`.

`kCGImageSourceCreateThumbnailFromImageAlways`
: Whether a thumbnail should be created from the full image even if a thumbnail is present in the image source file. The thumbnail is created from the full image, subject to the limit specified by `kCGImageSourceThumbnailMaxPixelSize`. If a maximum pixel size isn't specified, then the thumbnail is the size of the full image, which probably isn't what you want. This key must be a CFBoolean value. The default value is `kCFBooleanFalse`. This key can be provided in the options dictionary that you can pass to the function `CGImageSourceCreateThumbnailAtIndex` (page 243).

  Available in Mac OS X v10.4 and later.

  Declared in `CGImageSource.h`.

`kCGImageSourceThumbnailMaxPixelSize`
: The maximum width and height in pixels of a thumbnail. If this key is not specified, the width and height of a thumbnail is not limited and thumbnails may be as big as the image itself. If present, this key must be a CFNumber value. This key can be provided in the options dictionary that you pass to the function `CGImageSourceCreateThumbnailAtIndex` (page 243).

  Available in Mac OS X v10.4 and later.

  Declared in `CGImageSource.h`.

`kCGImageSourceCreateThumbnailWithTransform`
: Whether the thumbnail should be rotated and scaled according to the orientation and pixel aspect ratio of the full image. The value of this key must be a CFBoolean value. The default value is `kCFBooleanFalse`.

  Available in Mac OS X v10.4 and later.

  Declared in `CGImageSource.h`.

Discussion

Except for `kCGImageSourceTypeIdentifierHint`, which you use when creating an image source, these constants specify options that you can set when creating an image from image source. Each constant is a key; you must supply the appropriate value when you add this option to the options dictionary.

Declared In

`CGImageSource.h`

CGLayer Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGLayer.h

Overview

CGLayer objects are useful for offscreen drawing and can be used in much the same way that a bitmap context can be used. In fact, a CGLayer object is a much better representation than a bitmap context.

Using CGLayer objects can improve performance, particularly when you need to capture a piece of drawing that you stamp repeatedly (using the same scale factor and orientation). Quartz can cache CGLayer objects to the video card, making drawing a CGLayer to a destination much faster than rendering the equivalent image constructed from a bitmap context.

A CGLayer object is created relative to a graphics context. Although layer uses this graphics context as a reference for initialization, you are not restricted to drawing the layer to this graphics context. You can draw the layer to other graphics contexts, although any limitations of the original context are imposed. For example, if you create a CGLayer object using a bitmap context, the layer is rendered as a bitmap when drawn to any other graphics context.

You can use a CGLayer when you want to apply a shadow to a group of objects (such as a group of circles) rather than to individual objects.

Use these layers in your code whenever you can, especially when:

- You need to reuse a filled or stroked shape.
- You are building a scene and at least some of it can be reused. Put the reusable drawing in its own CGLayer.

Any CG object that you draw repeatedly—including CGPath, CGShading, and CGPDFPage—benefit from improved performance if you draw it to a CGLayer object.

Functions by Task

Creating Layer Objects

`CGLayerCreateWithContext` (page 255)
Creates a CGLayer object that is associated with a graphics context.

Drawing Layer Content

`CGContextDrawLayerInRect` (page 255)
Draws the contents of a CGLayer object into the specified rectangle.

`CGContextDrawLayerAtPoint` (page 254)
Draws the contents of a CGLayer object at the specified point.

Retaining and Releasing Layers

`CGLayerRelease` (page 257)
Decrements the retain count of a CGLayer object.

`CGLayerRetain` (page 258)
Increments the retain count of a CGLayer object.

Getting the CFType ID for a Layer

`CGLayerGetTypeID` (page 257)
Returns the unique type identifier used for CGLayer objects.

Getting Layer Information

`CGLayerGetSize` (page 257)
Returns the width and height of a CGLayer object.

`CGLayerGetContext` (page 256)
Returns the graphics context associated with a CGLayer object.

Functions

CGContextDrawLayerAtPoint

Draws the contents of a CGLayer object at the specified point.

```
void CGContextDrawLayerAtPoint (
   CGContextRef context,
   CGPoint point,
   CGLayerRef layer
);
```

Parameters

context
The graphics context associated with the layer.

point
The location, in current user space coordinates, to use as the origin for the drawing.

layer
The layer whose contents you want to draw.

Discussion
Calling the function `CGContextDrawLayerAtPoint` is equivalent to calling the function `CGContextDrawLayerInRect` with a rectangle that has its origin at `point` and its size equal to the size of the layer.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGLayer.h`

CGContextDrawLayerInRect

Draws the contents of a CGLayer object into the specified rectangle.

```
void CGContextDrawLayerInRect (
   CGContextRef context,
   CGRect rect,
   CGLayerRef layer
);
```

Parameters
context
The graphics context associated with the layer.

rect
The rectangle, in current user space coordinates, to draw to.

layer
The layer whose contents you want to draw.

Discussion
The contents are scaled, if necessary, to fit into the rectangle.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGLayer.h`

CGLayerCreateWithContext

Creates a CGLayer object that is associated with a graphics context.

```
CGLayerRef CGLayerCreateWithContext (
    CGContextRef context,
    CGSize size,
    CFDictionaryRef auxiliaryInfo
);
```

Parameters

context

The graphics context you want to create the layer relative to. The layer uses this graphics context as a reference for initialization.

size

The size, in default user space units, of the layer relative to the graphics context.

auxiliaryInfo

Reserved for future use. Pass `NULL`.

Return Value

A CGLayer object. You are responsible for releasing this object using the function `CGLayerRelease` (page 257) when you no longer need the layer.

Discussion

After you create a CGLayer object, you should reuse it whenever you can to facilitate the Quartz caching strategy. Quartz caches any objects that are reused, including CGLayer objects. Objects that are reused frequently remain in the cache. In contrast, objects that are used once in a while may be moved in and out of the cache according to their frequency of use. If you don't reuse CGLayer objects, Quartz won't cache them. This means that you lose an opportunity to improve the performance of your application.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGLayer.h`

CGLayerGetContext

Returns the graphics context associated with a CGLayer object.

```
CGContextRef CGLayerGetContext (
    CGLayerRef layer
);
```

Parameters

layer

The layer whose graphics context you want to obtain.

Return Value

The graphics context associated with the layer.

Discussion

The context that's returned is the context for the layer itself, not the context that you specified when you created the layer.

Availability

Available in Mac OS X version 10.4 and later.

Declared In
CGLayer.h

CGLayerGetSize

Returns the width and height of a CGLayer object.

```
CGSize CGLayerGetSize (
    CGLayerRef layer
);
```

Parameters

layer
The layer whose width and height you want to obtain.

Return Value
The width and height of the layer, in default user space coordinates.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGLayer.h

CGLayerGetTypeID

Returns the unique type identifier used for CGLayer objects.

```
CFTypeID CGLayerGetTypeID (
    void
);
```

Return Value
The type identifier for CGLayer objects.

Discussion
A type identifier is an integer that identifies the opaque type to which a Core Foundation object belongs. You use type IDs in various contexts, such as when you are operating on heterogeneous collections.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGLayer.h

CGLayerRelease

Decrements the retain count of a CGLayer object.

```
void CGLayerRelease (
    CGLayerRef layer
);
```

Parameters

layer

The layer to release.

Discussion

This function is equivalent to calling `CFRelease (layer)` except that it does not crash (as `CFRetain` does) if the `layer` parameter is `null`.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGLayer.h`

CGLayerRetain

Increments the retain count of a CGLayer object.

```
CGLayerRef CGLayerRetain (
    CGLayerRef layer
);
```

Parameters

layer

The layer to retain.

Return Value

The same layer you passed in as the `layer` parameter.

Discussion

This function is equivalent to calling `CFRetain (layer)` except that it does not crash (as `CFRetain` does) if the `layer` parameter is `null`.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGLayer.h`

Data Types

CGLayerRef

An opaque type used for offscreen drawing.

```
typedef struct CGLayer *CGLayerRef;
```

Availability

Available in Mac OS X v10.4 and later.

Declared In
`CGLayer.h`

CGPath Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPath.h
Companion guide	Quartz 2D Programming Guide

Overview

A **graphics path** is a description of a 2D geometric scene using sequences of lines and Bézier curves. `CGPathRef` defines an opaque type that represents an immutable graphics path. `CGMutablePathRef` defines an opaque type that represents a mutable graphics path. To draw using a Quartz path, you need to add the path to a graphics context—see `CGContextAddPath` (page 73).

Each figure in a scene may be described by a **subpath**. A subpath has an ordered set of **path elements**, that represent single steps in the construction of a subpath. (For example, `MoveToPoint (bottom left)` and `AddLineToPoint (bottom right)` are path elements.) A subpath also maintains state information, including a **starting point** and a **current point**. When drawing a path, Quartz traverses each subpath using its path elements and its state.

The lines and curves in a subpath are always connected, but they do not necessarily form a closed figure. Furthermore, subpaths do not need to be connected to each other. For example, you could use a graphics path to draw the outlines of a sequence of text characters.

Functions by Task

Creating and Managing Paths

`CGPathCreateMutable` (page 273)
Creates a mutable graphics path.

`CGPathCreateMutableCopy` (page 273)
Creates a mutable copy of an existing graphics path.

`CGPathCreateCopy` (page 272)
Creates an immutable copy of a graphics path.

`CGPathRelease` (page 277)
Decrements the retain count of a graphics path.

`CGPathRetain` (page 277)
Increments the retain count of a graphics path.

Modifying Quartz Paths

`CGPathAddArc` (page 263)
Appends an arc to a mutable graphics path, possibly preceded by a straight line segment.

`CGPathAddArcToPoint` (page 264)
Appends an arc to a mutable graphics path, possibly preceded by a straight line segment.

`CGPathAddCurveToPoint` (page 265)
Appends a Bézier curve to a mutable graphics path.

`CGPathAddLines` (page 267)
Appends an array of new line segments to a mutable graphics path.

`CGPathAddLineToPoint` (page 267)
Appends a line segment to a mutable graphics path.

`CGPathAddPath` (page 268)
Appends a path to a mutable graphics path.

`CGPathAddQuadCurveToPoint` (page 268)
Appends a quadratic curve to a mutable graphics path.

`CGPathAddRect` (page 269)
Appends a rectangle to a mutable graphics path.

`CGPathAddRects` (page 270)
Appends an array of rectangles to a mutable graphics path.

`CGPathApply` (page 271)
For each element in a graphics path, calls a custom applier function.

`CGPathMoveToPoint` (page 276)
Starts a new subpath at a specified location in a mutable graphics path.

`CGPathCloseSubpath` (page 271)
Closes and completes a subpath in a mutable graphics path.

`CGPathAddEllipseInRect` (page 266)
Adds to a path an ellipse that fits inside a rectangle.

Getting Information about Quartz Paths

`CGPathEqualToPath` (page 274)
Indicates whether two graphics paths are equivalent.

`CGPathGetBoundingBox` (page 274)
Returns the bounding box of a graphics path.

`CGPathGetCurrentPoint` (page 274)
Returns the current point in a graphics path.

`CGPathGetTypeID` (page 275)
Returns the Core Foundation type identifier for Quartz graphics paths.

`CGPathIsEmpty` (page 275)
Indicates whether or not a graphics path is empty.

`CGPathIsRect` (page 276)
Indicates whether or not a graphics path represents a rectangle.

`CGPathContainsPoint` (page 272)
Checks whether a point is contained in a graphics path.

Functions

CGPathAddArc

Appends an arc to a mutable graphics path, possibly preceded by a straight line segment.

```
void CGPathAddArc (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGFloat x,
   CGFloat y,
   CGFloat radius,
   CGFloat startAngle,
   CGFloat endAngle,
   bool clockwise
);
```

Parameters

path
The mutable graphics path to change.

m
A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the arc before it is added to the path.

x
The x-coordinate of the center point of the arc.

y
The y-coordinate of the center point of the arc.

r
The radius of the arc.

startAngle
The angle (in radians) from horizontal that determines the starting point of the arc.

endAngle
The angle (in radians) from horizontal that determines the ending point of the arc.

clockwise
A Boolean value that specifies whether or not to draw the arc in the clockwise direction; `true` specifies clockwise.

Discussion
An arc is a segment of a circle with radius r centered at a point `(x,y)`. When you call this function, you provide the center point, radius, and two angles in radians. Quartz uses this information to determine the end points of the arc, and then approximates the new arc using a sequence of cubic Bézier curves. The `clockwise` parameter determines the direction in which the arc is drawn.

A transformation may be applied to the Bézier curves before they are added to the path. If no transform is needed, the second argument should be `NULL`.

If the specified path already contains a subpath, Quartz implicitly adds a line connecting the current point to the beginning of the arc. If the path is empty, Quartz creates a new subpath for the arc and does not add the initial straight line segment.

The ending point of the arc becomes the new current point of the path.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathAddArcToPoint

Appends an arc to a mutable graphics path, possibly preceded by a straight line segment.

```
void CGPathAddArcToPoint (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGFloat x1,
   CGFloat y1,
   CGFloat x2,
   CGFloat y2,
   CGFloat radius
);
```

Parameters

path

The mutable path to change. The path must not be empty.

m

A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the arc before it is added to the path.

x1

The x-coordinate of the user space for the end point of the first tangent line. The first tangent line is drawn from the current point to `(x1,y1)`.

y1

The y-coordinate of the user space for the end point of the first tangent line. The first tangent line is drawn from the current point to `(x1,y1)`.

x2

The x-coordinate of the user space for the end point of the second tangent line. The second tangent line is drawn from `(x1,y1)` to `(x2,y2)`.

y2

The y-coordinate of the user space for the end point of the second tangent line. The second tangent line is drawn from `(x1,y1)` to `(x2,y2)`.

radius

The radius of the arc, in user space coordinates.

Discussion

This function uses a sequence of cubic Bézier curves to draw an arc that is tangent to the line from the current point to (x1,y1) and to the line from (x1,y1) to (x2,y2). The start and end points of the arc are located on the first and second tangent lines, respectively. The start and end points of the arc are also the "tangent points" of the lines.

If the current point and the first tangent point of the arc (the starting point) are not equal, Quartz appends a straight line segment from the current point to the first tangent point. After adding the arc, the current point is reset to the end point of the arc (the second tangent point).

For another way to draw an arc in a path, see `CGPathAddArc` (page 263).

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGPath.h`

CGPathAddCurveToPoint

Appends a Bézier curve to a mutable graphics path.

```
void CGPathAddCurveToPoint (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGFloat cp1x,
   CGFloat cp1y,
   CGFloat cp2x,
   CGFloat cp2y,
   CGFloat x,
   CGFloat y
);
```

Parameters

path

The mutable path to change. The path must not be empty.

m

A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the curve before it is added to the path.

cx1

The x-coordinate of the first control point.

cy1

The y-coordinate of the first control point.

cx2

The x-coordinate of the second control point.

cy2
The y-coordinate of the second control point.

x
The x-coordinate of the end point of the curve.

y
The y-coordinate of the end point of the curve.

Discussion
Appends a cubic Bézier curve from the current point in a path to the specified location using two control points, after an optional transformation. Before returning, this function updates the current point to the specified location `(x,y)`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathAddEllipseInRect

Adds to a path an ellipse that fits inside a rectangle.

```
void CGPathAddEllipseInRect (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGRect rect
);
```

Parameters

path
The path to modify.

m
An affine transform to apply to the ellipse, or `NULL` if you don't want to transform the ellipse.

rect
A rectangle to enclose the ellipse.

Discussion
The ellipse is approximated by a sequence of Bézier curves. Its center is the midpoint of the rectangle defined by the `rect` parameter. If the rectangle is square, then the ellipse is circular with a radius equal to one-half the width (or height) of the rectangle. If the `rect` parameter specifies a rectangular shape, then the major and minor axes of the ellipse are defined by the width and height of the rectangle.

The ellipse forms a complete subpath of the path—that is, the ellipse drawing starts with a move-to operation and ends with a close-subpath operation, with all moves oriented in the clockwise direction. If you supply an affine transform, then the constructed Bézier curves that define the ellipse are transformed before they are added to the path.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPath.h`

CGPathAddLines

Appends an array of new line segments to a mutable graphics path.

```
void CGPathAddLines (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   const CGPoint points[],
   size_t count
);
```

Parameters

path

The mutable path to change.

m

A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the lines before adding them to the path.

points

An array of points that specifies the line segments to add.

count

The number of elements in the array.

Discussion

This is a convenience function that adds a sequence of connected line segments to a path, using the following operation:

```
CGPathMoveToPoint (path, m, points[0].x, points[0].y);
for (k = 1; k < count; k++) {
    CGPathAddLineToPoint (path, m, points[k].x, points[k].y);
}
```

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGPath.h`

CGPathAddLineToPoint

Appends a line segment to a mutable graphics path.

```
void CGPathAddLineToPoint (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGFloat x,
   CGFloat y
);
```

Parameters

path

The mutable path to change. The path must not be empty.

m

A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the line before it is added to the path.

x
The x-coordinate of the end point of the line.

y
The y-coordinate of the end point of the line.

Discussion
Before returning, this function updates the current point to the specified location `(x,y)`.

Availability
Available in Mac OS X version 10.2 and later.

Related Sample Code
CALayerEssentials

Declared In
`CGPath.h`

CGPathAddPath

Appends a path to a mutable graphics path.

```
void CGPathAddPath (
   CGMutablePathRef path1,
   const CGAffineTransform *m,
   CGPathRef path2
);
```

Parameters

path1
The mutable path to change.

m
A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to `path2` before it is added to `path1`.

path2
The path to add.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathAddQuadCurveToPoint

Appends a quadratic curve to a mutable graphics path.

```
void CGPathAddQuadCurveToPoint (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGFloat cpx,
   CGFloat cpy,
   CGFloat x,
   CGFloat y
);
```

Parameters

path

The mutable path to change. The path must not be empty.

m

A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the curve before adding it to the path.

cx

The x-coordinate of the control point.

cy

The y-coordinate of the control point.

x

The x-coordinate of the end point of the curve.

y

The y-coordinate of the end point of the curve.

Discussion

Before returning, this function updates the current point to the specified location `(x, y)`.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGPath.h`

CGPathAddRect

Appends a rectangle to a mutable graphics path.

```
void CGPathAddRect (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGRect rect
);
```

Parameters

path

The mutable path to change.

m

A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the rectangle before adding it to the path.

rect

The rectangle to add.

Discussion
This is a convenience function that adds a rectangle to a path, using the following sequence of operations:

```
// start at origin
CGPathMoveToPoint (path, m, CGRectGetMinX(rect), CGRectGetMinY(rect));

// add bottom edge
CGPathAddLineToPoint (path, m, CGRectGetMaxX(rect), CGRectGetMinY(rect));

// add right edge
CGPathAddLineToPoint (path, m, CGRectGetMaxX(rect), CGRectGetMaxY(rect);

// add top edge
CGPathAddLineToPoint (path, m, CGRectGetMinX(rect), CGRectGetMaxY(rect));

// add left edge and close
CGPathCloseSubpath (path);
```

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathAddRects

Appends an array of rectangles to a mutable graphics path.

```
void CGPathAddRects (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   const CGRect rects[],
   size_t count
);
```

Parameters

path
The mutable path to change.

m
An affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the rectangles before adding them to the path.

rects
The array of new rectangles to add.

count
The number of elements in the array.

Discussion
This is a convenience function that adds an array of rectangles to a path, using the following operation:

```
for (k = 0; k < count; k++) {
    CGPathAddRect (path, m, rects[k]);
}
```

Availability
Available in Mac OS X version 10.2 and later.

Declared In
CGPath.h

CGPathApply

For each element in a graphics path, calls a custom applier function.

```
void CGPathApply (
   CGPathRef path,
   void *info,
   CGPathApplierFunction function
);
```

Parameters

path
The path to which the function will be applied.

info
A pointer to the user data that Quartz will pass to the function being applied, or NULL.

function
A pointer to the function to apply. See CGPathApplierFunction (page 278) for more information.

Discussion
For each element in the specified path, Quartz calls the applier function, which can examine (but not modify) the element.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
CGPath.h

CGPathCloseSubpath

Closes and completes a subpath in a mutable graphics path.

```
void CGPathCloseSubpath (
   CGMutablePathRef path
);
```

Parameters

path
The path to change.

Discussion
Appends a line from the current point in a path to the starting point of the current subpath and ends the subpath. On return, the current point is now the previous starting point.

Availability
Available in Mac OS X version 10.2 and later.

Related Sample Code
CALayerEssentials

Declared In
CGPath.h

CGPathContainsPoint

Checks whether a point is contained in a graphics path.

```
bool CGPathContainsPoint (
   CGPathRef path,
   const CGAffineTransform *m,
   CGPoint point,
   bool eoFill
);
```

Parameters

path

The path to evaluate the point against.

m

An affine transform. If `m` is not `NULL` then the point is transformed by this affine transform prior to determining whether the path contains the point.

point

The point to check.

eoFill

A Boolean value that, if `true`, specifies to use the even-odd fill rule to evaluate the painted region of the path. If `false`, the winding fill rule is used.

Return Value
Returns `true` if the point is contained in the path; `false` otherwise.

Discussion
A point is contained in a path if it is inside the painted region when the path is filled and the path is a closed path. You can call the function `CGPathCloseSubpath` to ensure that a path is closed.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGPath.h

CGPathCreateCopy

Creates an immutable copy of a graphics path.

```
CGPathRef CGPathCreateCopy (
   CGPathRef path
);
```

Parameters

path

The path to copy.

Return Value
A new, immutable copy of the specified path. You are responsible for releasing this object.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
CGPath.h

CGPathCreateMutable

Creates a mutable graphics path.

```
CGMutablePathRef CGPathCreateMutable (
   void
);
```

Return Value
A new mutable path. You are responsible for releasing this object.

Availability
Available in Mac OS X version 10.2 and later.

Related Sample Code
CALayerEssentials

Declared In
CGPath.h

CGPathCreateMutableCopy

Creates a mutable copy of an existing graphics path.

```
CGMutablePathRef CGPathCreateMutableCopy (
   CGPathRef path
);
```

Parameters

path
The path to copy.

Return Value
A new, mutable, copy of the specified path. You are responsible for releasing this object.

Discussion
You can modify a mutable graphics path by calling the various CGPath geometry functions, such as CGPathAddArc (page 263), CGPathAddLineToPoint (page 267), and CGPathMoveToPoint (page 276).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
CGPath.h

CGPathEqualToPath

Indicates whether two graphics paths are equivalent.

```
bool CGPathEqualToPath (
   CGPathRef path1,
   CGPathRef path2
);
```

Parameters

path1

The first path being compared.

path2

The second path being compared.

Return Value

A Boolean value that indicates whether or not the two specified paths contain the same sequence of path elements. If the paths are not the same, returns `false`.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGPath.h`

CGPathGetBoundingBox

Returns the bounding box of a graphics path.

```
CGRect CGPathGetBoundingBox (
   CGPathRef path
);
```

Parameters

path

The graphics path to evaluate.

Return Value

A rectangle that represents the bounding box of the specified path.

Discussion

The bounding box is the smallest rectangle completely enclosing all points in the path, including control points for Bézier and quadratic curves.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGPath.h`

CGPathGetCurrentPoint

Returns the current point in a graphics path.

```
CGPoint CGPathGetCurrentPoint (
   CGPathRef path
);
```

Parameters

path

The path to evaluate.

Return Value
The current point in the specified path.

Discussion
If the path is empty—that is, if it has no elements—this function returns `CGPointZero` (see `CGGeometry`). To determine whether a path is empty, use `CGPathIsEmpty` (page 275).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathGetTypeID

Returns the Core Foundation type identifier for Quartz graphics paths.

```
CFTypeID CGPathGetTypeID (
   void
);
```

Return Value
The Core Foundation identifier for the opaque type `CGPathRef` (page 278).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathIsEmpty

Indicates whether or not a graphics path is empty.

```
bool CGPathIsEmpty (
   CGPathRef path
);
```

Parameters

path

The path to evaluate.

Return Value
A Boolean value that indicates whether the specified path is empty.

Discussion
An empty path contains no elements.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathIsRect

Indicates whether or not a graphics path represents a rectangle.

```
bool CGPathIsRect (
   CGPathRef path,
   CGRect *rect
);
```

Parameters

path

The path to evaluate.

rect

On input, a pointer to an uninitialized rectangle. If the specified path represents a rectangle, on return contains a copy of the rectangle.

Return Value
A Boolean value that indicates whether the specified path represents a rectangle. If the path represents a rectangle, returns `true`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

CGPathMoveToPoint

Starts a new subpath at a specified location in a mutable graphics path.

```
void CGPathMoveToPoint (
   CGMutablePathRef path,
   const CGAffineTransform *m,
   CGFloat x,
   CGFloat y
);
```

Parameters

path

The mutable path to change.

m

A pointer to an affine transformation matrix, or `NULL` if no transformation is needed. If specified, Quartz applies the transformation to the point before changing the path.

x

The x-coordinate of the new location.

y
The y-coordinate of the new location.

Discussion
This function initializes the starting point and the current point to the specified location (x,y) after an optional transformation.

Availability
Available in Mac OS X version 10.2 and later.

Related Sample Code
CALayerEssentials

Declared In
`CGPath.h`

CGPathRelease

Decrements the retain count of a graphics path.

```
void CGPathRelease (
   CGPathRef path
);
```

Parameters
path
The graphics path to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `path` parameter is `NULL`.

Availability
Available in Mac OS X version 10.2 and later.

Related Sample Code
CALayerEssentials

Declared In
`CGPath.h`

CGPathRetain

Increments the retain count of a graphics path.

```
CGPathRef CGPathRetain (
   CGPathRef path
);
```

Parameters
path
The graphics path to retain.

Return Value
The same path you passed in as the `path` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `path` parameter is `NULL`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPath.h`

Callbacks

CGPathApplierFunction

Defines a callback function that can view an element in a graphics path.

```
typedef void (*CGPathApplierFunction) (
    void *info,
    const CGPathElement *element
);
```

If you name your function `MyCGPathApplierFunc`, you would declare it like this:

```
void MyCGPathApplierFunc (
   void *info,
  const CGPathElement *element
);
```

Discussion
See also `CGPathApply` (page 271).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGPath.h`

Data Types

CGPathRef

An opaque type that represents an immutable graphics path.

```
typedef const struct CGPath *CGPathRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGPath.h

CGMutablePathRef

An opaque type that represents a mutable graphics path.

```
typedef struct CGPath *CGMutablePathRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGPath.h

CGPathElement

A data structure that provides information about a path element.

```
struct CGPathElement {
    CGPathElementType type;
    CGPoint * points;
};
typedef struct CGPathElement CGPathElement;
```

Fields
`type`
An element type (or operation).
`points`
An array of one or more points that serve as arguments.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGPath.h

Constants

Path Drawing Modes

Options for rendering a path.

```
enum CGPathDrawingMode {
    kCGPathFill,
    kCGPathEOFill,
    kCGPathStroke,
    kCGPathFillStroke,
    kCGPathEOFillStroke
};
typedef enum CGPathDrawingMode CGPathDrawingMode;
```

Constants

`kCGPathFill`
Render the area contained within the path using the non-zero winding number rule.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGPathEOFill`
Render the area within the path using the even-odd rule.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGPathStroke`
Render a line along the path.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGPathFillStroke`
First fill and then stroke the path, using the nonzero winding number rule.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

`kCGPathEOFillStroke`
First fill and then stroke the path, using the even-odd rule.
Available in Mac OS X v10.0 and later.
Declared in `CGContext.h`.

Discussion
You can pass a path drawing mode constant to the function `CGContextDrawPath` (page 87) to specify how Quartz should paint a graphics context's current path.

Path Element Types

The type of element found in a path.

```
enum CGPathElementType {
    kCGPathElementMoveToPoint,
    kCGPathElementAddLineToPoint,
    kCGPathElementAddQuadCurveToPoint,
    kCGPathElementAddCurveToPoint,
    kCGPathElementCloseSubpath
};
typedef enum CGPathElementType CGPathElementType;
```

Constants

`kCGPathElementMoveToPoint`
The path element that starts a new subpath. See the function `CGPathMoveToPoint` (page 276).

Available in Mac OS X v10.2 and later.

Declared in `CGPath.h`.

`kCGPathElementAddLineToPoint`
The path element that adds a line from the current point to the specified point. See the function `CGPathAddLineToPoint` (page 267).

Available in Mac OS X v10.2 and later.

Declared in `CGPath.h`.

`kCGPathElementAddQuadCurveToPoint`
The path element that adds a quadratic curve from the current point to the specified point. See the function `CGPathAddQuadCurveToPoint` (page 268).

Available in Mac OS X v10.2 and later.

Declared in `CGPath.h`.

`kCGPathElementAddCurveToPoint`
The path element that adds a cubic curve from the current point to the specified point. See the function `CGPathAddCurveToPoint` (page 265).

Available in Mac OS X v10.2 and later.

Declared in `CGPath.h`.

`kCGPathElementCloseSubpath`
The path element that closes and completes a subpath. See the function `CGPathCloseSubpath` (page 271).

Available in Mac OS X v10.2 and later.

Declared in `CGPath.h`.

Discussion
For more information about paths, see `CGPathRef` (page 278).

CGPattern Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPattern.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPatternRef` opaque type represents a pattern that you can use to stroke along or fill in a graphics path. Quartz tiles the pattern cell for you, based on parameters you specify when you call `CGPatternCreate` (page 284).

To create a dashed line, see `CGContextSetLineDash` (page 117) in *CGContext Reference*.

Functions by Task

Creating a Pattern

`CGPatternCreate` (page 284)
Creates a pattern object.

Getting the CFType ID

`CGPatternGetTypeID` (page 285)
Returns the type identifier for Quartz patterns.

Retaining and Releasing a Pattern

`CGPatternRetain` (page 286)
Increments the retain count of a Quartz pattern.

`CGPatternRelease` (page 285)
Decrements the retain count of a Quartz pattern.

Functions

CGPatternCreate

Creates a pattern object.

```
CGPatternRef CGPatternCreate (
   void *info,
   CGRect bounds,
   CGAffineTransform matrix,
   CGFloat xStep,
   CGFloat yStep,
   CGPatternTiling tiling,
   bool isColored,
   const CGPatternCallbacks *callbacks
);
```

Parameters

info

A pointer to private storage used by your pattern drawing function, or `NULL`. For more information, see the discussion below.

bounds

The bounding box of the pattern cell, specified in pattern space. (Pattern space is an abstract space that maps to the default user space by the transformation matrix you specify with the `matrix` parameter.) The drawing done in your pattern drawing function is clipped to this rectangle.

matrix

A matrix that represents a transform from pattern space to the default user space of the context in which the pattern is used. If no transform is needed, pass the identity matrix.

xStep

The horizontal displacement between cells, specified in pattern space. For no additional horizontal space between cells (so that each pattern cells abuts the previous pattern cell in the horizontal direction), pass the width of the pattern cell.

yStep

The vertical displacement between cells, specified in pattern space. For no additional vertical space between cells (so that each pattern cells abuts the previous pattern cell in the vertical direction), pass the height of the pattern cell.

tiling

A `CGPatternTiling` constant that specifies the desired tiling method. For more information about tiling methods, see "Tiling Patterns" (page 289).

isColored

If you want to draw your pattern using its own intrinsic color, pass `true`. If you want to draw an uncolored (or masking) pattern that uses the fill or stroke color in the graphics state, pass `false`.

callbacks

A pointer to a pattern callback function table—your pattern drawing function is an entry in this table. See `CGPatternCallbacks` (page 288) for more information about callback function tables for patterns.

Return Value

A new Quartz pattern. You are responsible for releasing this object using `CGPatternRelease` (page 285).

Discussion
Quartz calls your drawing function at the appropriate time to draw the pattern cell. A pattern cell must be invariant—that is, the pattern cell should be drawn exactly the same way each time the drawing function is called.

The appearance of a pattern cell is unaffected by changes in the graphics state of the context in which the pattern is used.

See `CGPatternDrawPatternCallback` (page 286) for more information about pattern drawing functions.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CGPattern.h`

CGPatternGetTypeID

Returns the type identifier for Quartz patterns.

```
CFTypeID CGPatternGetTypeID (
    void
);
```

Return Value
The identifier for the opaque type `CGPatternRef` (page 288).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPattern.h`

CGPatternRelease

Decrements the retain count of a Quartz pattern.

```
void CGPatternRelease (
   CGPatternRef pattern
);
```

Parameters
pattern
The pattern to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the *pattern* parameter is `NULL`.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CGPattern.h`

CGPatternRetain

Increments the retain count of a Quartz pattern.

```
CGPatternRef CGPatternRetain (
   CGPatternRef pattern
);
```

Parameters

pattern

The pattern to retain.

Return Value

The same pattern you passed in as the `pattern` parameter.

Discussion

This function is equivalent to `CFRetain`, except that it does not cause an error if the `pattern` parameter is `NULL`.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CGPattern.h`

Callbacks

CGPatternDrawPatternCallback

Draws a pattern cell.

```
typedef void (*CGPatternDrawPatternCallback) (
   void * info,
    CGContextRef context
);
```

If you name your function `MyDrawPattern`, you would declare it like this:

```
void MyDrawPattern (
    void * info,
    CGContextRef context
);
```

Parameters

info

A generic pointer to private data associated with the pattern. This is the same pointer you supplied to `CGPatternCreate` (page 284).

context

The graphics context for drawing the pattern cell.

Discussion
When a pattern is used to stroke or fill a graphics path, Quartz calls your custom drawing function at the appropriate time to draw the pattern cell. The cell should be drawn exactly the same way each time the drawing function is called.

In a drawing function associated with an uncolored pattern, you should not attempt to set a stroke or fill color or color space—if you do so, the result is undefined.

To learn how to associate your drawing function with a Quartz pattern, see `CGPatternCreate` (page 284) and `CGPatternCallbacks` (page 288).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGPattern.h`

CGPatternReleaseInfoCallback

Release private data or resources associated with the pattern.

```
typedef void (*CGPatternReleaseInfoCallback) (
    void * info
);
```

If you name your function `MyCGPatternReleaseInfo`, you would declare it like this:

```
void MyCGPatternReleaseInfo (
    void * info
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPatternCreate` (page 284).

Discussion
Quartz calls your release function when it frees your pattern object.

To learn how to associate your release function with a Quartz pattern, see `CGPatternCreate` (page 284) and `CGPatternCallbacks` (page 288).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGPattern.h`

Data Types

CGPatternRef

An opaque type that represents a pattern.

```
typedef struct CGPattern * CGPatternRef;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CGPattern.h`

CGPatternCallbacks

A structure that holds a version and two callback functions for drawing a custom pattern.

```
struct CGPatternCallbacks {
    unsigned int version;
    CGPatternDrawPatternCallback drawPattern;
    CGPatternReleaseInfoCallback releaseInfo;
};
typedef struct CGPatternCallbacks CGPatternCallbacks;
```

Fields

`version`
: The version of the structure passed in as a parameter to the `CGPatternCreate` (page 284). For this version of the structure, you should set this value to zero.

`drawPattern`
: A pointer to a custom function that draws the pattern. For information about this callback function, see `CGPatternDrawPatternCallback` (page 286).

`releaseInfo`
: An optional pointer to a custom function that's invoked when the pattern is released. `CGPatternReleaseInfoCallback` (page 287).

Discussion
You supply a `CGPatternCallbacks` structure to the function `CGPatternCreate` (page 284) to create a data provider for direct access. The functions specified by the `CGPatternCallbacks` structure are responsible for drawing the pattern and for handling the pattern's memory management.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CGPattern.h`

Constants

Tiling Patterns

Different methods for rendering a tiled pattern.

```
enum CGPatternTiling {
    kCGPatternTilingNoDistortion,
    kCGPatternTilingConstantSpacingMinimalDistortion,
    kCGPatternTilingConstantSpacing
};
typedef enum CGPatternTiling CGPatternTiling;
```

Constants

`kCGPatternTilingNoDistortion`

The pattern cell is not distorted when painted. The spacing between pattern cells may vary by as much as 1 device pixel.

Available in Mac OS X v10.1 and later.

Declared in `CGPattern.h`.

`kCGPatternTilingConstantSpacingMinimalDistortion`

Pattern cells are spaced consistently. The pattern cell may be distorted by as much as 1 device pixel when the pattern is painted.

Available in Mac OS X v10.1 and later.

Declared in `CGPattern.h`.

`kCGPatternTilingConstantSpacing`

Pattern cells are spaced consistently, as with `kCGPatternTilingConstantSpacingMinimalDistortion`. The pattern cell may be distorted additionally to permit a more efficient implementation.

Available in Mac OS X v10.1 and later.

Declared in `CGPattern.h`.

Declared In

`CGPattern.h`

CGPDFArray Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFArray.h
Companion guide	Quartz 2D Programming Guide

Overview

The CGPDFArray header file defines an opaque type that encapsulates a PDF array. A PDF array represents an array structure in a PDF document. PDF arrays may be heterogeneous—that is, they may contain any other PDF objects, including PDF strings, PDF dictionaries, and other PDF arrays.

Many CGPDFArray functions to retrieve values from a PDF array take the form:

```
bool CGPDFArrayGet<DataType> (
 CGPDFArrayRef array,
 size_t index,
 <DataType>Ref *value
);
```

These functions test the data type of the object at the specified index. If the object is not of the expected type, the function returns `false`. If the object is of the expected type, the function returns `true`, and the object is passed back in the `value` parameter.

This opaque type is not derived from CFType and therefore there are no functions for retaining and releasing it. CGPDFArray objects exist only as constituent parts of a CGPDFDocument object, and they are managed by their container.

Functions

CGPDFArrayGetArray

Returns whether an object at a given index in a PDF array is another PDF array and, if so, retrieves that array.

```
bool CGPDFArrayGetArray (
   CGPDFArrayRef array,
   size_t index,
   CGPDFArrayRef *value
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index

The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value

On input, a pointer to a PDF array. If the value at the specified index is a PDF array, then on return that array, otherwise the value is unspecified.

Return Value
Returns `true` if there is a PDF array at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFArray.h`

CGPDFArrayGetBoolean

Returns whether an object at a given index in a PDF array is a PDF Boolean and, if so, retrieves that Boolean.

```
bool CGPDFArrayGetBoolean (
   CGPDFArrayRef array,
   size_t index,
   CGPDFBoolean *value
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index

The index of the value to retrieve. If the index is outside the index space of `array` (`0` to `N-1`, where `N` is the count of `array`), the behavior is undefined.

value

On input, a pointer to a PDF Boolean. If the value at the specified index is a PDF Boolean, then on return that Boolean, otherwise the value is undefined.

Return Value
Returns `true` if there is a PDF Boolean at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFArray.h`

CGPDFArrayGetCount

Returns the number of items in a PDF array.

```
size_t CGPDFArrayGetCount (
   CGPDFArrayRef array
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

Return Value

Returns the number of items in the array.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFArray.h`

CGPDFArrayGetDictionary

Returns whether an object at a given index in a PDF array is a PDF dictionary and, if so, retrieves that dictionary.

```
bool CGPDFArrayGetDictionary (
   CGPDFArrayRef array,
   size_t index,
   CGPDFDictionaryRef *value
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index

The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value

On input, a pointer to a PDF dictionary. If the value at the specified index is a PDF dictionary, then on return that dictionary, otherwise the value is undefined.

Return Value

Returns `true` if there is a PDF dictionary at the specified index, otherwise `false`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFArray.h`

CGPDFArrayGetInteger

Returns whether an object at a given index in a PDF array is a PDF integer and, if so, retrieves that object.

```
bool CGPDFArrayGetInteger (
   CGPDFArrayRef array,
   size_t index,
   CGPDFInteger *value
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index

The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value

On input, a pointer to a PDF integer. If the value at the specified index is a PDF integer value, then on return contains that value, otherwise the value is undefined.

Return Value
Returns `true` if there is a PDF integer at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFArray.h`

CGPDFArrayGetName

Returns whether an object at a given index in a PDF array is a PDF name reference (represented as a constant C string) and, if so, retrieves that name.

```
bool CGPDFArrayGetName (
   CGPDFArrayRef array,
   size_t index,
   const char **value
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index

The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value

An uninitialized pointer to a constant C string. If the value at the specified index is a reference to a PDF name (represented by a constant C string) then upon return, contains that value; otherwise the value is undefined.

Return Value
Returns `true` if there is an array of characters at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CGPDFArray.h

CGPDFArrayGetNull

Returns whether an object at a given index in a Quartz PDF array is a PDF null.

```
bool CGPDFArrayGetNull (
   CGPDFArrayRef array,
   size_t index
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index

The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

Return Value
Returns `true` if there is a PDF null at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CGPDFArray.h

CGPDFArrayGetNumber

Returns whether an object at a given index in a PDF array is a PDF number and, if so, retrieves that object.

```
bool CGPDFArrayGetNumber (
   CGPDFArrayRef array,
   size_t index,
   CGPDFReal *value
);
```

Parameters

array

A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index

The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value

On input, a pointer to a PDF number. If the value at the specified index is a PDF number, then on return contains that value, otherwise the value is undefined.

Return Value
Returns `true` if there is a PDF number at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CGPDFArray.h

CGPDFArrayGetObject

Returns whether an object at a given index in a PDF array is a PDF object and, if so, retrieves that object.

```
bool CGPDFArrayGetObject (
   CGPDFArrayRef array,
   size_t index,
   CGPDFObjectRef *value
);
```

Parameters

array
A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index
The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value
On input, a pointer to a PDF object. If the value at the specified index is a PDF object, then on return contains that object, otherwise the value is undefined.

Return Value
Returns `true` if there is a PDF object at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CGPDFArray.h

CGPDFArrayGetStream

Returns whether an object at a given index in a PDF array is a PDF stream and, if so, retrieves that stream.

```
bool CGPDFArrayGetStream (
   CGPDFArrayRef array,
   size_t index,
   CGPDFStreamRef *value
);
```

Parameters

array
A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index
The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value
On input, a pointer to a PDF stream. If the value at the specified index is a PDF stream, then on return that stream, otherwise the value is undefined.

Return Value
Returns `true` if there is a PDF stream at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFArray.h`

CGPDFArrayGetString

Returns whether an object at a given index in a PDF array is a PDF string and, if so, retrieves that string.

```
bool CGPDFArrayGetString (
   CGPDFArrayRef array,
   size_t index,
   CGPDFStringRef *value
);
```

Parameters

array
A PDF array. If this parameter is not a valid PDF array, the behavior is undefined.

index
The index of the value to retrieve. If the index is outside the index space of the array (`0` to `N-1`, where `N` is the count of the array), the behavior is undefined.

value
On input, a pointer to a PDF string. If the value at the specified index is a PDF string, then on return that string, otherwise the value is undefined.

Return Value
Returns `true` if there is a PDF stream at the specified index, otherwise `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFArray.h`

Data Types

CGPDFArrayRef

An opaque type that encapsulates a PDF array.

```
typedef struct CGPDFArray *CGPDFArrayRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPDFArray.h`

CGPDFContentStream Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFContentStream.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFContentStreamRef` opaque type provides access to the data that describes the appearance of a PDF page. A CGPDFContentStream object represents one or more PDF content streams for a page and their associated resource dictionaries. A PDF content stream is a sequential set of instructions that specifies how to paint items on a PDF page. A resource dictionary contains information needed by the content stream in order to decode the sequential instructions of the content stream.

CGPDFContentStream functions can retrieve both the content streams and the resource dictionaries associated with a PDF page.

This opaque type is not derived from CFType and therefore there are no functions for retaining and releasing it.

Functions by Task

Creating a PDF Content Stream Object

`CGPDFContentStreamCreateWithPage` (page 300)
Creates a content stream object from a PDF page object.

`CGPDFContentStreamCreateWithStream` (page 300)
Creates a PDF content stream object from an existing PDF content stream object.

Getting Data from a PDF Content Stream Object

`CGPDFContentStreamGetStreams` (page 301)
Gets the array of PDF content streams contained in a PDF content stream object.

`CGPDFContentStreamGetResource` (page 301)
Gets the specified resource from a PDF content stream object.

Retaining and Releasing a PDF Content Stream Object

`CGPDFContentStreamRetain` (page 302)
Increments the retain count of a PDF content stream object.

`CGPDFContentStreamRelease` (page 302)
Decrements the retain count of a PDF content stream object.

Functions

CGPDFContentStreamCreateWithPage

Creates a content stream object from a PDF page object.

```
CGPDFContentStreamRef CGPDFContentStreamCreateWithPage (
   CGPDFPageRef page
);
```

Parameters

page
A PDF page object.

Return Value
A new CGPDFContentStream object. You are responsible for releasing this object by calling the function `CGPDFContentStreamRelease`.

Discussion
A CGPDFContentStream object can contain more than one PDF content stream. To retrieve an array of the PDF content streams in the object, call the function `CGPDFContentStreamGetStreams` (page 301). To obtain the resources associated with a CGPDFContentStream object, call the function `CGPDFContentStreamGetResource` (page 301).

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFContentStream.h`

CGPDFContentStreamCreateWithStream

Creates a PDF content stream object from an existing PDF content stream object.

```
CGPDFContentStreamRef CGPDFContentStreamCreateWithStream (
   CGPDFStreamRef stream,
   CGPDFDictionaryRef streamResources,
   CGPDFContentStreamRef parent
);
```

Parameters

stream
The PDF stream you want to create a content stream from.

streamResources
A PDF dictionary that contains the resources associated with the stream you want to retrieve.

parent
The content stream of the page on which *stream* appears. Supply the `parent` parameter when you create a content stream that's used within a page.

Return Value
A CGPDFContentStream object created from the `stream` parameter. You are responsible for releasing this object by calling the function `CGPDFContentStreamRelease` (page 302).

Discussion
You can use this function to get access to the contents of a form, pattern, Type3 font, or any PDF stream.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFContentStream.h`

CGPDFContentStreamGetResource

Gets the specified resource from a PDF content stream object.

```
CGPDFObjectRef CGPDFContentStreamGetResource (
   CGPDFContentStreamRef cs,
   const char *category,
   const char *name
);
```

Parameters

cs
A CGPDFContentStream object.

category
A string that specifies the category of the resource you want to obtain.

name
A string that specifies the name of the resource you want to obtain.

Return Value
The resource dictionary.

Discussion
You can use this function to obtain resources used by the content stream, such as forms, patterns, color spaces, and fonts.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFContentStream.h`

CGPDFContentStreamGetStreams

Gets the array of PDF content streams contained in a PDF content stream object.

```
CFArrayRef CGPDFContentStreamGetStreams (
   CGPDFContentStreamRef cs
);
```

Parameters

cs

A CGPDFContentStream object.

Return Value
The array of PDF content streams that make up the content stream object represented by the `cs` parameter.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFContentStream.h`

CGPDFContentStreamRelease

Decrements the retain count of a PDF content stream object.

```
void CGPDFContentStreamRelease (
   CGPDFContentStreamRef cs
);
```

Parameters

cs

A PDF content stream.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFContentStream.h`

CGPDFContentStreamRetain

Increments the retain count of a PDF content stream object.

```
CGPDFContentStreamRef CGPDFContentStreamRetain (
   CGPDFContentStreamRef cs
);
```

Parameters

cs

A PDF content stream.

Return Value
The same PDF content stream you passed in as the `cs` parameter.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFContentStream.h`

Data Types

CGPDFContentStreamRef

An opaque type that provides access to the data that describes the appearance of a PDF page.

```
typedef struct CGPDFContentStream *CGPDFContentStreamRef;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFContentStream.h`

CGPDFContext Reference

Derived From:	`CGContextRef` (page 137)
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFContext.h
Companion guide	Quartz 2D Programming Guide

Overview

The CGPDFContext header file defines functions that create and get information about a Quartz PDF context. A CGPDFContext object is a type of `CGContextRef` (page 137) that is used for drawing PDF content. The functions in this reference operate only on Quartz PDF graphics contexts created using the functions CGPDFContextCreate (page 307) or CGPDFContextCreateWithURL (page 308).

When you draw to the PDF context using CGContext functions the drawing operations are recorded in PDF format. The PDF commands that represent the drawing are written to the destination specified when you create the PDF graphics context.

Functions by Task

Creating a Context

`CGPDFContextCreate` (page 307)
Creates a PDF graphics context.

`CGPDFContextCreateWithURL` (page 308)
Creates a URL-based PDF graphics context.

Beginning and Ending Pages

`CGPDFContextBeginPage` (page 306)
Begins a new page in a PDF graphics context.

`CGPDFContextEndPage` (page 309)
Ends the current page in the PDF graphics context.

Working with Destinations

`CGPDFContextAddDestinationAtPoint` (page 306)
Sets a destination to jump to when a point in the current page of a PDF graphics context is clicked.

`CGPDFContextSetDestinationForRect` (page 309)
Sets a destination to jump to when a rectangle in the current PDF page is clicked.

`CGPDFContextSetURLForRect` (page 310)
Sets the URL associated with a rectangle in a PDF graphics context.

Closing a PDF Context

`CGPDFContextClose` (page 307)
Closes a PDF document.

Functions

CGPDFContextAddDestinationAtPoint

Sets a destination to jump to when a point in the current page of a PDF graphics context is clicked.

```
void CGPDFContextAddDestinationAtPoint (
    CGContextRef context,
    CFStringRef name,
    CGPoint point
);
```

Parameters

context
A PDF graphics context.

name
A destination name.

point
A location in the current page of the PDF graphics context.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFContext.h`

CGPDFContextBeginPage

Begins a new page in a PDF graphics context.

```
void CGPDFContextBeginPage (
    CGContextRef context,
    CFDictionaryRef pageInfo
);
```

Parameters

context

A PDF graphics context.

pageInfo

A dictionary that contains key-value pairs that define the page properties.

Discussion

You must call the function `CGPDFContextEndPage` (page 309) to signal the end of the page.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGPDFContext.h`

CGPDFContextClose

Closes a PDF document.

```
void CGPDFContextClose(
    CGContextRef context
);
```

Parameters

context

A PDF graphics context.

Discussion

After closing the context, all pending data is written to the context destination, and the PDF file is completed. No additional data can be written to the destination context after the PDF document is closed.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CGPDFContext.h`

CGPDFContextCreate

Creates a PDF graphics context.

```
CGContextRef CGPDFContextCreate (
   CGDataConsumerRef consumer,
   const CGRect *mediaBox,
   CFDictionaryRef auxiliaryInfo
);
```

Parameters

consumer

The data consumer to receive the PDF output data.

mediaBox

A pointer to a rectangle that defines the size and location of the PDF page, or `NULL`. The origin of the rectangle should typically be `(0,0)`. Quartz uses this rectangle as the default bounds of the page's media box. If you pass `NULL`, Quartz uses a default page size of 8.5 by 11 inches (612 by 792 points).

auxiliaryInfo

A dictionary that specifies any additional information to be used by the PDF context when generating the PDF file, or `NULL`. The dictionary is retained by the new context, so on return you may safely release it. See "Auxiliary Dictionary Keys" (page 310) for keys you can include in the dictionary.

Return Value

A new PDF context, or `NULL` if the context cannot be created. You are responsible for releasing this object using `CGContextRelease` (page 102).

Discussion

This function creates a PDF drawing environment to your specifications. When you draw into the new context, Quartz renders your drawing as a sequence of PDF drawing commands that are passed to the data consumer object.

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CGPDFContext.h`

CGPDFContextCreateWithURL

Creates a URL-based PDF graphics context.

```
CGContextRef CGPDFContextCreateWithURL (
   CFURLRef url,
   const CGRect *mediaBox,
   CFDictionaryRef auxiliaryInfo
);
```

Parameters

url

A Core Foundation URL that specifies where you want to place the resulting PDF file.

mediaBox

A rectangle that specifies the bounds of the PDF. The origin of the rectangle should typically be `(0,0)`. The `CGPDFContextCreateWithURL` function uses this rectangle as the default page media bounding box. If you pass `NULL`, `CGPDFContextCreateWithURL` uses a default page size of 8.5 by 11 inches (612 by 792 points).

auxiliaryInfo

A dictionary that specifies any additional information to be used by the PDF context when generating the PDF file, or `NULL`. The dictionary is retained by the new context, so on return you may safely release it.

Return Value
A new PDF context, or `NULL` if a context could not be created. You are responsible for releasing this object using `CGContextRelease` (page 102).

Discussion
When you call this function, Quartz creates a PDF drawing environment—that is, a graphics context—to your specifications. When you draw into the resulting context, Quartz renders your drawing as a series of PDF drawing commands stored in the specified location.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGPDFContext.h`

CGPDFContextEndPage

Ends the current page in the PDF graphics context.

```
void CGPDFContextEndPage (
    CGContextRef context
);
```

Parameters

context

A PDF graphics context.

Discussion
You can call `CGPDFContextEndPage` only after you call the function `CGPDFContextBeginPage` (page 306).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFContext.h`

CGPDFContextSetDestinationForRect

Sets a destination to jump to when a rectangle in the current PDF page is clicked.

```
void CGPDFContextSetDestinationForRect (
    CGContextRef context,
    CFStringRef name,
    CGRect rect
);
```

Parameters

context

A PDF graphics context.

name

A destination name.

rect

A rectangle that specifies an area of the current page of a PDF graphics context. The rectangle is specified in default user space (not device space).

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGPDFContext.h

CGPDFContextSetURLForRect

Sets the URL associated with a rectangle in a PDF graphics context.

```
void CGPDFContextSetURLForRect (
   CGContextRef context,
   CFURLRef url,
   CGRect rect
);
```

Parameters

context

A PDF graphics context.

url

A CFURL object that specifies the destination of the contents associated with the rectangle.

rect

A rectangle specified in default user space (not device space).

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGPDFContext.h

Constants

Auxiliary Dictionary Keys

Keys that used to set up a PDF context.

```
CFStringRef kCGPDFContextAuthor;
CFStringRef kCGPDFContextCreator;
CFStringRef kCGPDFContextTitle;
CFStringRef kCGPDFContextOwnerPassword;
CFStringRef kCGPDFContextUserPassword;
CFStringRef kCGPDFContextAllowsPrinting;
CFStringRef kCGPDFContextAllowsCopying;
CFStringRef kCGPDFContextOutputIntent;
CFStringRef kCGPDFContextOutputIntents;
CFStringRef kCGPDFContextSubject;
CFStringRef kCGPDFContextKeywords;
CFStringRef kCGPDFContextEncryptionKeyLength;
```

Constants

`kCGPDFContextAuthor`

The corresponding value is a string that represents the name of the person who created the document. This key is optional.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextCreator`

The corresponding value is a string that represents the name of the application used to produce the document. This key is optional.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextTitle`

The corresponding value is a string that represents the title of the document. This key is optional.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextOwnerPassword`

The owner password of the PDF document. If this key is specified, the document is encrypted using the value as the owner password; otherwise, the document will not be encrypted. The value of this key must be a CFString object that can be represented in ASCII encoding. Only the first 32 bytes are used for the password. There is no default value for this key. If the value of this key cannot be represented in ASCII, the document is not created and the creation function returns `NULL`.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextUserPassword`

The user password of the PDF document. If the document is encrypted, then the value of this key will be the user password for the document. If not specified, the user password is the empty string. The value of this key must be a CFString object that can be represented in ASCII encoding; only the first 32 bytes will be used for the password. If the value of this key cannot be represented in ASCII, the document is not created and the creation function returns `NULL`.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextAllowsPrinting`

Whether the document allows printing when unlocked with the user password. The value of this key must be a CFBoolean value. The default value of this key is `kCFBooleanTrue`.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextAllowsCopying`
Whether the document allows copying when unlocked with the user password. The value of this key must be a CFBoolean object. The default value of this key is `kCFBooleanTrue`.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextOutputIntent`
The output intent `PDF/X`. This key is optional. If present, the value of this key must be a CFDictionary object. The dictionary is added to the `/OutputIntents` entry in the PDF file document catalog. The keys and values contained in the dictionary must match those specified in section 9.10.4 of the PDF 1.4 specification, ISO/DIS 15930-3 document published by ISO/TC 130, and Adobe Technical Note #5413.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextOutputIntents`
Output intent dictionaries. This key is optional. If present, the value must be an array of one or more `kCGPDFContextOutputIntent` dictionaries. The array is added to the PDF document in the `/OutputIntents` entry in the PDF file's document catalog. Each dictionary in the array must be of form specified for the `kCGPDFContextOutputIntent` key, except that only the first dictionary in the array is required to contain the "S" key with a value of `GTS_PDFX`. If both the `kCGPDFContextOutputIntent` and `kCGPDFContextOutputIntents` keys are specified, the former is ignored.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextSubject`
The subject of a document. Optional; if present, the value of this key must be a `CFString` object.

Declared in `CGPDFContext.h`.

Available in Mac OS X v10.5 and later.

`kCGPDFContextKeywords`
The keywords for this document. This key is optional. If the value of this key is a `CFString` object, the `/Keywords` entry will be the specified string. If the value of this key is a `CFArray` object, then it must be an array of `CFString` objects. The `/Keywords` entry will, in this case, be the concatenation of the specified strings separated by commas (" , "). In addition, an entry with the key `"/AAPL:Keywords"` is stored in the document information dictionary; its value is an array consisting of each of the specified strings. The value of this key must be in one of the above forms; otherwise, this key is ignored.

Declared in `CGPDFContext.h`.

Available in Mac OS X v10.5 and later.

`kCGPDFContextEncryptionKeyLength`
The encryption key length in bits; see Table 3.18 "Entries common to all encryption dictionaries", PDF Reference: Adobe PDF version 1.5 (4th ed.) for more information. Optional; if present, the value of this key must be a `CFNumber` object with value which is a multiple of 8 between 40 and 128, inclusive. If this key is absent or invalid, the encryption key length defaults to 40 bits.

Declared in `CGPDFContext.h`.

Available in Mac OS X v10.5 and later.

Discussion
For more information about using these keys in a PDF context, see `CGPDFContextCreate` (page 307) and `CGPDFContextCreateWithURL` (page 308).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFContext.h`

Box Dictionary Keys

Keys that specify various PDF boxes.

```
CFStringRef kCGPDFContextMediaBox
CFStringRef kCGPDFContextCropBox
CFStringRef kCGPDFContextBleedBox
CFStringRef kCGPDFContextTrimBox
CFStringRef kCGPDFContextArtBox
```

Constants

`kCGPDFContextMediaBox`
The media box for the document or for a given page. This key is optional. If present, the value of this key must be a CFData object that contains a `CGRect` (stored by value, not by reference).

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextCropBox`
The crop box for the document or for a given page. This key is optional. If present, the value of this key must be a CFData object that contains a `CGRect` (stored by value, not by reference).

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextBleedBox`
The bleed box for the document or for a given page. This key is optional. If present, the value of this key must be a CFData object that contains a `CGRect` (stored by value, not by reference).

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextTrimBox`
The trim box for the document or for a given page. This key is optional. If present, the value of this key must be a CFData object that contains a `CGRect` (stored by value, not by reference).

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFContextArtBox`
The art box for the document or for a given page. This key is optional. If present, the value of this key must be a CFData object that contains a `CGRect` (stored by value, not by reference).

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

Discussion
For more information about using these keys in a PDF context, see `CGPDFContextCreate` (page 307) and `CGPDFContextCreateWithURL` (page 308).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFContext.h`

Output Intent Dictionary Keys

Keys to specify output intent options.

```
CFStringRef kCGPDFXOutputIntentSubtype;
CFStringRef kCGPDFXOutputConditionIdentifier;
CFStringRef kCGPDFXOutputCondition;
CFStringRef kCGPDFXRegistryName;
CFStringRef kCGPDFXInfo;
CFStringRef kCGPDFXDestinationOutputProfile;
```

Constants

`kCGPDFXOutputIntentSubtype`
The output intent subtype. This key is required. The value of this key must be a CFString object equal to `"GTS_PDFX"`; otherwise, the dictionary is ignored.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFXOutputConditionIdentifier`
A string identifying the intended output device or production condition in a human- or machine-readable form. This key is required. The value of this key must be a CFString object. For best results, the string should be restricted to characters in the ASCII character set.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFXOutputCondition`
A text string identifying the intended output device or production condition in a human- readable form. This key is optional. If present, the value of this key must be a CFString object.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFXRegistryName`
A string identifying the registry in which the condition designated by `kCGPDFXOutputConditionIdentifier` is defined. This key is optional. If present, the value of this key must be a CFString object. For best results, the string should be lossless in ASCII encoding.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFXInfo`
A human-readable text string containing additional information or comments about the intended target device or production condition. This key is required if the value of `kCGPDFXOutputConditionIdentifier` does not specify a standard production condition. It is optional otherwise. If present, the value of this key must be a CFString object.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

`kCGPDFXDestinationOutputProfile`

An ICC profile stream defining the transformation from the PDF document's source colors to output device colorants. This key is required if the value of `kCGPDFXOutputConditionIdentifier` does not specify a standard production condition. It is optional otherwise. If present, the value of this key must be an ICC-based color space specified as a `CGColorSpace` object.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFContext.h`.

Discussion
For more information about using these keys in a PDF context, see `CGPDFContextCreate` (page 307) and `CGPDFContextCreateWithURL` (page 308).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFContext.h`

CGPDFDictionary Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFDictionary.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFDictionaryRef` opaque type encapsulates a PDF dictionary whose key-value pairs can specify any kind of PDF object, including another dictionary. Dictionary objects are the main building blocks of a PDF document. A key-value pair within a dictionary is called an entry. In a PDF dictionary, the key must be an array of characters. Within a given dictionary, the keys are unique—that is, no two keys in a single dictionary are equal (as determined by `strcmp`). The value associated with a key can be any kind of PDF object, including another dictionary. Dictionary objects are the main building blocks of a PDF document.

Many functions that retrieve values from a PDF dictionary take the form:

```
bool CGPDFDictionaryGet<DataType> (
 CGPDFDictionaryRef dictionary,
 const char *key,
 <DataType>Ref *value
);
```

These functions test whether there is an object associated with the specified key. If there is an object associated with the specified key, they test its data type. If there is no associated object, or if there is but it is not of the expected type, the function returns `false`. If there is an object associated with the specified key and it is of the expected type, the function returns `true` and the object is passed back in the `value` parameter.

This opaque type is not derived from CFType and therefore there are no functions for retaining and releasing it. CGPDFDictionary objects exist only as constituent parts of a CGPDFDocument object, and they are managed by their container.

Functions by Task

Applying a Function to All Entries

`CGPDFDictionaryApplyFunction` (page 318)
Applies a function to each entry in a dictionary.

Getting Data from a Dictionary

`CGPDFDictionaryGetArray` (page 319)
Returns whether there is a PDF array associated with a specified key in a PDF dictionary and, if so, retrieves that array.

`CGPDFDictionaryGetBoolean` (page 320)
Returns whether there is a PDF Boolean value associated with a specified key in a PDF dictionary and, if so, retrieves the Boolean value.

`CGPDFDictionaryGetCount` (page 320)
Returns the number of entries in a PDF dictionary.

`CGPDFDictionaryGetDictionary` (page 320)
Returns whether there is another PDF dictionary associated with a specified key in a PDF dictionary and, if so, retrieves that dictionary.

`CGPDFDictionaryGetInteger` (page 321)
Returns whether there is a PDF integer associated with a specified key in a PDF dictionary and, if so, retrieves that integer.

`CGPDFDictionaryGetName` (page 322)
Returns whether an object with a specified key in a PDF dictionary is a PDF name reference (represented as a constant C string) and, if so, retrieves that name.

`CGPDFDictionaryGetNumber` (page 322)
Returns whether there is a PDF number associated with a specified key in a PDF dictionary and, if so, retrieves that number.

`CGPDFDictionaryGetObject` (page 323)
Returns whether there is a PDF object associated with a specified key in a PDF dictionary and, if so, retrieves that object.

`CGPDFDictionaryGetStream` (page 323)
Returns whether there is a PDF stream associated with a specified key in a PDF dictionary and, if so, retrieves that stream.

`CGPDFDictionaryGetString` (page 324)
Returns whether there is a PDF string associated with a specified key in a PDF dictionary and, if so, retrieves that string.

Functions

CGPDFDictionaryApplyFunction

Applies a function to each entry in a dictionary.

```
void CGPDFDictionaryApplyFunction (
   CGPDFDictionaryRef dict,
   CGPDFDictionaryApplierFunction function,
   void *info
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

function

The function to apply to each entry in the dictionary.

info

A pointer to contextual information to pass to the function.

Discussion

This function enumerates all of the entries in the dictionary, calling the function once for each. The current key, its associated value, and the contextual information are passed to the function (see also CGPDFDictionaryApplierFunction (page 324)).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDictionaryGetArray

Returns whether there is a PDF array associated with a specified key in a PDF dictionary and, if so, retrieves that array.

```
bool CGPDFDictionaryGetArray (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFArrayRef *value
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key

The key for the value to retrieve.

value

On input, an uninitialized pointer to a PDF array. If the value associated with the specified key is a PDF array, then on return contains that array; otherwise the value is unspecified.

Return Value

Returns `true` if there is a PDF array associated with the specified key; otherwise, `false`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDictionaryGetBoolean

Returns whether there is a PDF Boolean value associated with a specified key in a PDF dictionary and, if so, retrieves the Boolean value.

```
bool CGPDFDictionaryGetBoolean (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFBoolean *value
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key

The key for the value to retrieve.

value

On input, a pointer to a PDF Boolean value. If the value associated with the specified key is a PDF Boolean value, then on return contains that value; otherwise the value is unspecified.

Return Value

Returns `true` if there is a PDF Boolean value associated with the specified key; otherwise, `false`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDictionaryGetCount

Returns the number of entries in a PDF dictionary.

```
size_t CGPDFDictionaryGetCount (
   CGPDFDictionaryRef dict
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

Return Value

Returns the number of entries in the dictionary.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDictionaryGetDictionary

Returns whether there is another PDF dictionary associated with a specified key in a PDF dictionary and, if so, retrieves that dictionary.

```
bool CGPDFDictionaryGetDictionary (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFDictionaryRef *value
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key

The key for the value to retrieve.

value

On input, a pointer to a PDF dictionary. If the value associated with the specified key is a PDF dictionary, then on return contains that dictionary; otherwise the value is unspecified.

Return Value

Returns `true` if there is a PDF dictionary associated with the specified key; otherwise, `false`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDictionaryGetInteger

Returns whether there is a PDF integer associated with a specified key in a PDF dictionary and, if so, retrieves that integer.

```
bool CGPDFDictionaryGetInteger (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFInteger *value
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key

The key for the value to retrieve.

value

On input, a pointer to a PDF integer. If the value associated with the specified key is a PDF integer, then on return contains that value; otherwise the value is unspecified.

Return Value

Returns `true` if there is a PDF integer associated with the specified key; otherwise, `false`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDictionaryGetName

Returns whether an object with a specified key in a PDF dictionary is a PDF name reference (represented as a constant C string) and, if so, retrieves that name.

```
bool CGPDFDictionaryGetName (
   CGPDFDictionaryRef dict,
   const char *key,
   const char **value
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key

The key for the value to retrieve.

value

On input, a pointer to a PDF name reference, represented as a constant C string. If the value associated with the specified key is a reference to a PDF name, then on return, the variable points to the name; otherwise, the value is undefined.

Return Value

Returns `true` if there is a character array associated with the specified key; otherwise, `false`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDictionaryGetNumber

Returns whether there is a PDF number associated with a specified key in a PDF dictionary and, if so, retrieves that number.

```
bool CGPDFDictionaryGetNumber (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFReal *value
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key

The key for the value to retrieve.

value

On input, a pointer to a PDF number. If the value associated with the specified key is a PDF number (real or integer), then on return contains that value; otherwise the value is unspecified.

Return Value

Returns `true` if there is a PDF number associated with the specified key; otherwise, `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFDictionary.h`

CGPDFDictionaryGetObject

Returns whether there is a PDF object associated with a specified key in a PDF dictionary and, if so, retrieves that object.

```
bool CGPDFDictionaryGetObject (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFObjectRef *value
);
```

Parameters

dictionary
A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key
The key for the value to retrieve.

value
On input, a pointer to a PDF object. If the value associated with the specified key is a PDF object, then on return contains that object; otherwise the value is unspecified.

Return Value
Returns `true` if there is a PDF object associated with the specified key; otherwise, `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFDictionary.h`

CGPDFDictionaryGetStream

Returns whether there is a PDF stream associated with a specified key in a PDF dictionary and, if so, retrieves that stream.

```
bool CGPDFDictionaryGetStream (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFStreamRef *value
);
```

Parameters

dictionary
A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key
The key for the value to be retrieved.

value

On input, a pointer to a PDF stream. If the value associated with the specified key is a PDF stream, then on return contains that stream; otherwise, the value is unspecified.

Return Value
Returns `true` if there is a PDF stream associated with the specified key; otherwise, `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFDictionary.h`

CGPDFDictionaryGetString

Returns whether there is a PDF string associated with a specified key in a PDF dictionary and, if so, retrieves that string.

```
bool CGPDFDictionaryGetString (
   CGPDFDictionaryRef dict,
   const char *key,
   CGPDFStringRef *value
);
```

Parameters

dictionary

A PDF dictionary. If this parameter is not a valid PDF dictionary, the behavior is undefined.

key

The key for the value to retrieve.

value

On input, a pointer to a PDF string. If the value associated with the specified key is a PDF string, then on return contains that string; otherwise the value is unspecified.

Return Value
Returns `true` if there is a PDF string associated with the specified key; otherwise, `false`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFDictionary.h`

Callbacks

CGPDFDictionaryApplierFunction

Performs custom processing on a key-value pair from a PDF dictionary, using optional contextual information.

```
typedef void (*CGPDFDictionaryApplierFunction) (
   const char *key,
   CGPDFObjectRef value,
   void *info,
);
```

If you name your function `MyFunction`, you would declare it like this:

```
void MyFunction (
   const char *key,
   CGPDFObjectRef object,
   void *info
);
```

Parameters

key

The current key in the dictionary.

object

The value in the dictionary associated with the key.

info

The contextual information that your provided in the `info` parameter in `CGPDFDictionaryApplyFunction` (page 318).

Discussion

`CGPDFDictionaryApplierFunction` defines the callback for `CGPDFDictionaryApplyFunction`, that enumerates all of the entries in the dictionary, calling your custom applier function once for each entry. The current key, its associated value, and the contextual information are passed to your applier function using the `key`, `value`, and `info` parameters respectively.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPDFDictionary.h`

Data Types

CGPDFDictionaryRef

An opaque type that encapsulates a PDF dictionary.

```
typedef struct CGPDFDictionary *CGPDFDictionaryRef;
```

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPDFDictionary.h`

CGPDFDocument Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFDocument.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFDocumentRef` opaque type encapsulates a document that contains PDF (Portable Document Format) drawing information. PDF provides an efficient format for cross-platform exchange of documents with rich content. PDF files can contain multiple pages of images and text. A PDF document object contains all the information relating to a PDF document, including its catalog and contents.

Note that PDF documents may be encrypted, and that some operations may be restricted until a valid password is supplied—see the functions listed in "Managing Encryption" (page 328). Quartz also supports decrypting encrypted documents.

Quartz can both display and generate files that are compliant with the PDF standard. When imaging PDF files, `CGPDFDocumentRef` is the basic type used to represent a PDF document.

Functions by Task

Creating PDF Document Objects

`CGPDFDocumentCreateWithProvider` (page 330)
Creates a Quartz PDF document using a data provider.

`CGPDFDocumentCreateWithURL` (page 330)
Creates a Quartz PDF document using data specified by a URL.

Retaining and Releasing PDF Documents

`CGPDFDocumentRelease` (page 339)
Decrements the retain count of a PDF document.

`CGPDFDocumentRetain` (page 339)
Increments the retain count of a Quartz PDF document.

Getting the CFType ID for a PDF Document Object

`CGPDFDocumentGetTypeID` (page 337)
Returns the type identifier for Quartz PDF documents.

Getting Information About Quartz PDF Documents

`CGPDFDocumentGetCatalog` (page 332)
Returns the document catalog of a Quartz PDF document.

`CGPDFDocumentGetNumberOfPages` (page 335)
Returns the number of pages in a PDF document.

`CGPDFDocumentGetPage` (page 335)
Returns a page from a Quartz PDF document.

`CGPDFDocumentGetVersion` (page 337)
Returns the major and minor version numbers of a Quartz PDF document.

`CGPDFDocumentGetInfo` (page 334)
Gets the information dictionary for a PDF document.

`CGPDFDocumentGetID` (page 333)
Gets the file identifier for a PDF document.

Managing Encryption

`CGPDFDocumentAllowsCopying` (page 329)
Returns whether the specified PDF document allows copying.

`CGPDFDocumentAllowsPrinting` (page 329)
Returns whether a PDF document allows printing.

`CGPDFDocumentIsEncrypted` (page 338)
Returns whether the specified PDF file is encrypted.

`CGPDFDocumentIsUnlocked` (page 338)
Returns whether the specified PDF document is currently unlocked.

`CGPDFDocumentUnlockWithPassword` (page 340)
Unlocks an encrypted PDF document, if a valid password is supplied.

Getting Page Information

`CGPDFDocumentGetArtBox` (page 331) Deprecated in Mac OS X version 10.3 and later
Returns the art box of a page in a PDF document.

`CGPDFDocumentGetBleedBox` (page 331) Deprecated in Mac OS X version 10.3 and later
Returns the bleed box of a page in a PDF document.

`CGPDFDocumentGetCropBox` (page 333) Deprecated in Mac OS X version 10.3 and later
Returns the crop box of a page in a PDF document.

`CGPDFDocumentGetMediaBox` (page 334) Deprecated in Mac OS X version 10.3 and later
Returns the media box of a page in a PDF document.

`CGPDFDocumentGetRotationAngle` (page 336) Deprecated in Mac OS X version 10.3 and later
Returns the rotation angle of a page in a PDF document.

`CGPDFDocumentGetTrimBox` (page 336) Deprecated in Mac OS X version 10.3 and later
Returns the trim box of a page in a PDF document.

Functions

CGPDFDocumentAllowsCopying

Returns whether the specified PDF document allows copying.

```
bool CGPDFDocumentAllowsCopying (
   CGPDFDocumentRef document
);
```

Parameters

document
A PDF document.

Return Value

A Boolean that, if `true`, indicates that the document allows copying. If the value is `false`, the document does not allow copying.

Discussion

This function returns `true` if the specified PDF document allows copying. It returns `false` if the document is encrypted and the current password doesn't grant permission to perform copying.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGPDFDocument.h`

CGPDFDocumentAllowsPrinting

Returns whether a PDF document allows printing.

```
bool CGPDFDocumentAllowsPrinting (
   CGPDFDocumentRef document
);
```

Parameters

document
A PDF document.

Return Value

A Boolean that, if `true`, indicates that the document allows printing. If the value is `false`, the document does not allow printing.

Discussion

This function returns `true` if the specified PDF document allows printing. It returns `false` if the document is encrypted and the current password doesn't grant permission to perform printing.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentCreateWithProvider

Creates a Quartz PDF document using a data provider.

```
CGPDFDocumentRef CGPDFDocumentCreateWithProvider (
   CGDataProviderRef provider
);
```

Parameters

provider

A data provider that supplies the PDF document data.

Return Value
A new Quartz PDF document, or `NULL` if a document can not be created. You are responsible for releasing the object using `CGPDFDocumentRelease` (page 339).

Discussion
Distributing individual pages of a PDF document to separate threads is not supported. If you want to use threads, consider creating a separate document for each thread and operating on a block of pages per thread.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextDrawPDFDocument` (page 88)

Related Sample Code
CarbonSketch

Declared In
`CGPDFDocument.h`

CGPDFDocumentCreateWithURL

Creates a Quartz PDF document using data specified by a URL.

```
CGPDFDocumentRef CGPDFDocumentCreateWithURL (
   CFURLRef url
);
```

Parameters

url

The URL address at which the PDF document data is located.

Return Value
A new Quartz PDF document, or `NULL` if a document could not be created. You are responsible for releasing the object using `CGPDFDocumentRelease` (page 339).

Discussion
Distributing individual pages of a PDF document to separate threads is not supported. If you want to use threads, consider creating a separate document for each thread and operating on a block of pages per thread.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGContextDrawPDFDocument` (page 88)

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetArtBox

Returns the art box of a page in a PDF document. (Deprecated in Mac OS X version 10.3 and later.)

```
CGRect CGPDFDocumentGetArtBox (
   CGPDFDocumentRef document,
   int page
);
```

Parameters

document
The PDF document to examine.

page
An integer that specifies the number of the page to examine.

Return Value
A rectangle that represents the art box for the specified page, expressed in default PDF user space units (points).

Discussion
The replacement function for this one is `CGPDFPageGetBoxRect`, which gets the rectangle associated with a type of box (art, media, crop, bleed trim) that represents a content region or page dimensions of a PDF page. For more information see *CGPDFPage Reference*.

The art box defines the extent of the page's meaningful content (including potential white space) as intended by the document creator. The default value is the page's crop box.

Availability
Available in Mac OS X version 10.0 and later.
Deprecated in Mac OS X version 10.3 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetBleedBox

Returns the bleed box of a page in a PDF document. (Deprecated in Mac OS X version 10.3 and later.)

```
CGRect CGPDFDocumentGetBleedBox (
   CGPDFDocumentRef document,
   int page
);
```

Parameters

document

The PDF document to examine.

page

An integer that specifies the number of the page to examine.

Return Value
A rectangle that represents the bleed box for the specified page, expressed in default PDF user space units (points).

Discussion
The replacement function for this one is `CGPDFPageGetBoxRect`, which gets the rectangle associated with a type of box (art, media, crop, bleed trim) that represents a content region or page dimensions of a PDF page. For more information see *CGPDFPage Reference*.

The bleed box defines the bounds to which the contents of the page should be clipped when output in a production environment. The default value is the page's crop box.

Availability
Available in Mac OS X version 10.0 and later.
Deprecated in Mac OS X version 10.3 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetCatalog

Returns the document catalog of a Quartz PDF document.

```
CGPDFDictionaryRef CGPDFDocumentGetCatalog (
   CGPDFDocumentRef document
);
```

Parameters

document

A PDF document.

Return Value
The document catalog of the specified document.

Discussion
The entries in a PDF document catalog recursively describe the contents of the PDF document. You can access the contents of a PDF document catalog by calling the function `CGPDFDocumentGetCatalog`. For information on accessing PDF metadata, see *Quartz 2D Programming Guide*.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetCropBox

Returns the crop box of a page in a PDF document. (Deprecated in Mac OS X version 10.3 and later.)

```
CGRect CGPDFDocumentGetCropBox (
   CGPDFDocumentRef document,
   int page
);
```

Parameters

document

The PDF document to examine.

page

An integer that specifies the number of the page to examine.

Return Value

A rectangle that represents the crop box for the specified page, expressed in default PDF user space units (points).

Discussion

The replacement function for this one is `CGPDFPageGetBoxRect`, which gets the rectangle associated with a type of box (art, media, crop, bleed trim) that represents a content region or page dimensions of a PDF page. For more information see *CGPDFPage Reference*.

The crop box defines the region to which the contents of the page are to be clipped (or cropped) when displayed or printed. Unlike the other boxes, the crop box has no defined meaning in terms of physical page geometry or intended use—it merely suggests where the page should be clipped.

Availability

Available in Mac OS X version 10.0 and later.

Deprecated in Mac OS X version 10.3 and later.

Declared In

`CGPDFDocument.h`

CGPDFDocumentGetID

Gets the file identifier for a PDF document.

```
CGPDFArrayRef CGPDFDocumentGetID (
   CGPDFDocumentRef document
);
```

Parameters

document

The document whose file identifier you want to obtain.

Return Value

Returns the file identifier for the document.

Discussion

A PDF file identifier is defined in the PDF specification as an array of two strings, the first of which is a permanent identifier that doesn't change even when the file is updated. The second string changes each time the file is updated. For more information, see *PDF Reference: Version 1.3 (Second Edition)*, Adobe Systems Incorporated.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetInfo

Gets the information dictionary for a PDF document.

```
CGPDFDictionaryRef CGPDFDocumentGetInfo (
   CGPDFDocumentRef document
);
```

Parameters

document
The document whose dictionary you want to obtain.

Return Value
The information dictionary for the document.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetMediaBox

Returns the media box of a page in a PDF document. (Deprecated in Mac OS X version 10.3 and later.)

```
CGRect CGPDFDocumentGetMediaBox (
   CGPDFDocumentRef document,
   int page
);
```

Parameters

document
The PDF document to examine.

page
An integer that specifies the number of the page to examine.

Return Value
A rectangle that represents the media box for the specified page, expressed in default PDF user space units (points).

Discussion
The replacement function for this one is `CGPDFPageGetBoxRect`, which gets the rectangle associated with a type of box (art, media, crop, bleed trim) that represents a content region or page dimensions of a PDF page. For more information see *CGPDFPage Reference*.

The media box defines the location and size of the physical medium on which the page is intended to be displayed or printed. For example, if the page size is 8.5 by 11 inches, this function returns the coordinate pairs `(0,0)` and `(612,792)`.

Availability
Available in Mac OS X version 10.0 and later.
Deprecated in Mac OS X version 10.3 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetNumberOfPages

Returns the number of pages in a PDF document.

```
size_t CGPDFDocumentGetNumberOfPages (
   CGPDFDocumentRef document
);
```

Parameters

document
The PDF document to examine.

Return Value
The total number of pages in the PDF document.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetPage

Returns a page from a Quartz PDF document.

```
CGPDFPageRef CGPDFDocumentGetPage (
   CGPDFDocumentRef document,
   size_t pageNumber
);
```

Parameters

document
A PDF document.

pageNumber
The number of the page requested.

Return Value
Return the PDF page corresponding to the specified page number, or `NULL` if no such page exists in the document. Pages are numbered starting at 1.

Availability
Available in Mac OS X version 10.3 and later.

Related Sample Code
CarbonSketch

Declared In
CGPDFDocument.h

CGPDFDocumentGetRotationAngle

Returns the rotation angle of a page in a PDF document. (Deprecated in Mac OS X version 10.3 and later.)

```
int CGPDFDocumentGetRotationAngle (
   CGPDFDocumentRef document,
   int page
);
```

Parameters

document

The PDF document to examine.

page

An integer that specifies the number of the page to examine.

Return Value
The rotation angle of the page, expressed in degrees. If the specified page does not exist, returns 0.

Discussion
The replacement function for this one is `CGPDFPageGetRotationAngle`. For more information see *CGPDFPage Reference*.

Availability
Available in Mac OS X version 10.0 and later.
Deprecated in Mac OS X version 10.3 and later.

Declared In
CGPDFDocument.h

CGPDFDocumentGetTrimBox

Returns the trim box of a page in a PDF document. (Deprecated in Mac OS X version 10.3 and later.)

```
CGRect CGPDFDocumentGetTrimBox (
   CGPDFDocumentRef document,
   int page
);
```

Parameters

document

The PDF document to examine.

page

A value specifying the number of the page to examine.

Return Value
Returns a rectangle that represents the trim box for the specified page, expressed in default PDF user space units (points).

Discussion
The replacement function for this one is `CGPDFPageGetBoxRect`, which gets the rectangle associated with a type of box (art, media, crop, bleed trim) that represents a content region or page dimensions of a PDF page. For more information see *CGPDFPage Reference*.

The trim box defines the intended dimensions of the finished page after trimming. It may be smaller than the media box, to allow for production-related content such as printing instructions, cut marks, or color bars. The default value is the page's crop box.

Availability
Available in Mac OS X version 10.0 and later.
Deprecated in Mac OS X version 10.3 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetTypeID

Returns the type identifier for Quartz PDF documents.

```
CFTypeID CGPDFDocumentGetTypeID (
   void
);
```

Return Value
The identifier for the opaque type `CGPDFDocumentRef` (page 340).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentGetVersion

Returns the major and minor version numbers of a Quartz PDF document.

```
void CGPDFDocumentGetVersion (
   CGPDFDocumentRef document,
   int *majorVersion,
   int *minorVersion
);
```

Parameters

document
A PDF document.

majorVersion
On return, contains the major version number of the document.

minorVersion
On return, contains the minor version number of the document.

Return Value
On return, the values of the `majorVersion` and `minorVersion` parameters are set to the major and minor version numbers of the document respectively.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentIsEncrypted

Returns whether the specified PDF file is encrypted.

```
bool CGPDFDocumentIsEncrypted (
   CGPDFDocumentRef document
);
```

Parameters

document
A PDF document.

Return Value
A Boolean that, if `true`, indicates that the document is encrypted. If the value is `false`, the document is not encrypted.

Discussion
If the document is encrypted, a password must be supplied before certain operations are enabled. For more information, see `CGPDFDocumentUnlockWithPassword` (page 340).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentIsUnlocked

Returns whether the specified PDF document is currently unlocked.

```
bool CGPDFDocumentIsUnlocked (
   CGPDFDocumentRef document
);
```

Parameters

document
A PDF document.

Return Value
A Boolean that, if `true`, indicates that the document is not locked. If the value is `false`, the document is locked.

Discussion
There are two possible reasons why a PDF document is unlocked:

- The document is not encrypted.
- The document is encrypted, and a valid password was previously specified using `CGPDFDocumentUnlockWithPassword` (page 340).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentRelease

Decrements the retain count of a PDF document.

```
void CGPDFDocumentRelease (
    CGPDFDocumentRef document
);
```

Parameters

document
The PDF document to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `document` parameter is `NULL`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentRetain

Increments the retain count of a Quartz PDF document.

```
CGPDFDocumentRef CGPDFDocumentRetain (
    CGPDFDocumentRef document
);
```

Parameters

document
The PDF document to retain.

Return Value
The same document you passed in as the *document* parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `document` parameter is `NULL`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGPDFDocument.h`

CGPDFDocumentUnlockWithPassword

Unlocks an encrypted PDF document, if a valid password is supplied.

```
bool CGPDFDocumentUnlockWithPassword (
   CGPDFDocumentRef document,
   const char *password
);
```

Parameters

document

A PDF document.

password

A pointer to a string that contains the password.

Return Value
A Boolean that, if `true`, indicates that the document has been successfully unlocked. If the value is `false`, the document has not been unlocked.

Discussion
Given an encrypted PDF document and a password, this function does the following:

- Sets the lock state of the document, based on the validity of the password.
- Returns `true` if the document is unlocked.
- Returns `false` if the document cannot be unlocked with the specified password.

Unlocking a PDF document makes it possible to decrypt the document and perform other privileged operations. Different passwords enable different operations.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGPDFDocument.h`

Data Types

CGPDFDocumentRef

An opaque type that represents a PDF (Portable Document Format) document.

```
typedef struct CGPDFDocument * CGPDFDocumentRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGPDFDocument.h`

CGPDFObject Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFObject.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFObjectRef` opaque type represents PDF objects in a PDF document. PDF supports several basic types of object: Boolean values, integer and real numbers, strings, names, arrays, dictionaries, and streams. Most of these are represented in Quartz by corresponding specific types. A CGPDFObject can represent any of these types. You use CGPDFObject functions to determine the type of the object, and retrieve the object value if it is of an expected type.

This opaque type is not derived from CFType and therefore there are no functions for retaining and releasing it. CGPDFObject objects exist as constituent parts of a CGPDFDocument object, and are managed by their container.

Functions

CGPDFObjectGetType

Returns the PDF type identifier of an object.

```
CGPDFObjectType CGPDFObjectGetType (
   CGPDFObjectRef object
);
```

Parameters

object

A PDF object. If the value if not a PDF object, the behavior is unspecified.

Return Value

Returns the type of the `object` parameter. See "Data Types" (page 344).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFObject.h`

CGPDFObjectGetValue

Returns whether an object is of a given type and if it is, retrieves its value.

```
bool CGPDFObjectGetValue (
   CGPDFObjectRef object,
   CGPDFObjectType type,
   void *value
);
```

Parameters

object

A PDF object.

type

A PDF object type.

value

If the `object` parameter is a PDF object of the specified type, then on return contains that object, otherwise the value is unspecified.

Return Value

Returns `true` if the specified object is a PDF object of the specified type, otherwise `false`.

Discussion

The function gets the value of the `object` parameter. If the type of `object` is equal to the type specified, then:

- If the `value` parameter is not a null pointer, then the value of `object` is copied to `value`, and the function returns `true`.
- If the `value` parameter is a null pointer, then the function simply returns `true`. This allows you to test whether `object` is of the type specified.

If the type of `object` is `kCGPDFObjectTypeInteger` and `type` is equal to `kCGPDFObjectTypeReal`, then the value of `object` is converted to floating point, the result copied to `value`, and the function returns `true`. If none of the preceding conditions is met, returns `false`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFObject.h`

Data Types

CGPDFObjectRef

An opaque type that contains information about a PDF object.

```
typedef union CGPDFObject *CGPDFObjectRef;
```

Availability

Available in Mac OS X v10.3 and later.

Declared In
CGPDFObject.h

CGPDFBoolean

A PDF Boolean value.

```
typedef unsigned char CGPDFBoolean;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGPDFObject.h

CGPDFInteger

A PDF integer value.

```
typedef long int CGPDFInteger;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGPDFObject.h

CGPDFReal

A PDF real value.

```
typedef CGFloat CGPDFReal;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGPDFObject.h

Constants

PDF Object Types

Types of PDF object.

```
enum CGPDFObjectType {
    kCGPDFObjectTypeNull = 1,
    kCGPDFObjectTypeBoolean,
    kCGPDFObjectTypeInteger,
    kCGPDFObjectTypeReal,
    kCGPDFObjectTypeName,
    kCGPDFObjectTypeString,
    kCGPDFObjectTypeArray,
    kCGPDFObjectTypeDictionary,
    kCGPDFObjectTypeStream
};typedef enum CGPDFObjectType CGPDFObjectType;
```

Constants

`kCGPDFObjectTypeNull`
The type for a PDF null.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeBoolean`
The type for a PDF Boolean.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeInteger`
The type for a PDF integer.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeReal`
The type for a PDF real.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeName`
Type for a PDF name.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeString`
The type for a PDF string.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeArray`
Type for a PDF array.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeDictionary`
The type for a PDF dictionary.
Available in Mac OS X v10.3 and later.
Declared in `CGPDFObject.h`.

`kCGPDFObjectTypeStream`
The type for a PDF stream.

Available in Mac OS X v10.3 and later.

Declared in `CGPDFObject.h`.

Declared In
`CGPDFObject.h`

CGPDFOperatorTable Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFOperatorTable.h

Overview

A CGPDFOperatorTable object stores callback functions for PDF operators. You pass an operator table and a PDF content stream to a CGPDFScanner object. When the scanner parses a PDF operator, Quartz invokes your callback for that operator. See also *CGPDFScanner Reference* and *CGPDFContentStream Reference*.

Note: This opaque type is not derived from CFType and therefore you can't use the Core Foundation base functions on it, such as `CFRetain` and `CFRelease`. Memory management is handled by the specific functions `CGPDFOperatorTableRetain` (page 350) and `CGPDFOperatorTableRelease` (page 350).

For more about PDF operators, see the latest version of *PDF Reference*, Adobe Systems Incorporated.

Functions by Task

Creating a PDF Operator Table

`CGPDFOperatorTableCreate` (page 350)
Creates an empty PDF operator table.

Setting Callback Functions

`CGPDFOperatorTableSetCallback` (page 351)
Sets a callback function for a PDF operator.

Retaining and Releasing a PDF Operator Table

`CGPDFOperatorTableRetain` (page 350)
Increments the retain count of a CGPDFOperatorTable object.

`CGPDFOperatorTableRelease` (page 350)
Decrements the retain count of a CGPDFOperatorTable object.

Functions

CGPDFOperatorTableCreate

Creates an empty PDF operator table.

```
CGPDFOperatorTableRef CGPDFOperatorTableCreate (
   void
);
```

Return Value
An empty PDF operator table. You are responsible for releasing this object by calling `CGPDFOperatorTableRelease` (page 350).

Discussion
Call the function `CGPDFOperatorTableSetCallback` (page 351) to fill the operator table with callbacks.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFOperatorTable.h`

CGPDFOperatorTableRelease

Decrements the retain count of a CGPDFOperatorTable object.

```
void CGPDFOperatorTableRelease (
   CGPDFOperatorTableRef table
);
```

Parameters
table
A PDF operator table.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFOperatorTable.h`

CGPDFOperatorTableRetain

Increments the retain count of a CGPDFOperatorTable object.

```
CGPDFOperatorTableRef CGPDFOperatorTableRetain (
   CGPDFOperatorTableRef table
);
```

Parameters

table

A PDF operator table.

Return Value

The same PDF operator table you passed in as the `table` parameter.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFOperatorTable.h`

CGPDFOperatorTableSetCallback

Sets a callback function for a PDF operator.

```
void CGPDFOperatorTableSetCallback (
   CGPDFOperatorTableRef table,
   const char *name,
   CGPDFOperatorCallback callback
);
```

Parameters

table

A PDF operator table.

name

The name of the PDF operator you want to set a callback for.

callback

The callback to invoke for the PDF operator specified by the `name` parameter.

Discussion

You call the function `CGPDFOperatorTableSetCallback` for each PDF operator for which you want to provide a callback. See Appendix A in the *PDF Reference, Second Edition*, version 1.3, Adobe Systems Incorporated for a summary of PDF operators.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFOperatorTable.h`

Callbacks

CGPDFOperatorCallback

Performs custom processing for PDF operators.

```
typedef void (*CGPDFOperatorCallback)(
   CGPDFScannerRef scanner,
   void *info
);
```

If you name your function `MyCGPDFOperatorCallback`, you would declare it like this:

```
void MyCGPDFOperatorCallback (
   CGPDFScannerRef scanner,
   void *info
);
```

Parameters

scanner

A CGPDFScanner object. Quartz passes the scanner to your callback function. The scanner contains the PDF content stream that has the PDF operator that corresponds to this callback.

info

A pointer to data passed to the callback.

Discussion

Your callback function takes any action that's appropriate for your application. For example, if you want to count the number of inline images in a PDF but ignore the image data, you would set a callback for the `EI` operator. In your callback you would increment a counter for each call.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGPDFOperatorTable.h`

Data Types

CGPDFOperatorTableRef

An opaque type that stores callback functions for PDF operators.

```
typedef struct CGPDFOperatorTable *CGPDFOperatorTableRef;
```

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGPDFOperatorTable.h`

CGPDFPage Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFPage.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFPageRef` opaque type represents a page in a PDF document.

Functions by Task

Retaining and Releasing a PDF Page

`CGPDFPageRetain` (page 358)
Increments the retain count of a PDF page.

`CGPDFPageRelease` (page 357)
Decrements the retain count of a PDF page.

Getting the CFType ID

`CGPDFPageGetTypeID` (page 357)
Returns the CFType ID for PDF page objects.

Getting Page Information

`CGPDFPageGetBoxRect` (page 354)
Returns the rectangle that represents a type of box for a content region or page dimensions of a PDF page.

`CGPDFPageGetDictionary` (page 354)
Returns the dictionary of a PDF page.

`CGPDFPageGetDocument` (page 355)
Returns the document for a page.

`CGPDFPageGetDrawingTransform` (page 355)
Returns the affine transform that maps a box to a given rectangle on a PDF page.

`CGPDFPageGetPageNumber` (page 356)
Returns the page number of the specified PDF page.

`CGPDFPageGetRotationAngle` (page 357)
Returns the rotation angle of a PDF page.

Functions

CGPDFPageGetBoxRect

Returns the rectangle that represents a type of box for a content region or page dimensions of a PDF page.

```
CGRect CGPDFPageGetBoxRect (
   CGPDFPageRef page,
   CGPDFBox box
);
```

Parameters

page
A PDF page.

box
A `CGPDFBox` constant that specifies the type of box. For possible values, see "PDF Boxes" (page 359).

Return Value

Returns the rectangle associated with the type of box specified by the `box` parameter in the specified page.

Discussion

Returns the rectangle associated with the specified box in the specified page. This is the value of the corresponding entry (such as `/MediaBox`, `/ArtBox`, and so on) in the page's dictionary.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CarbonSketch

Declared In

`CGPDFPage.h`

CGPDFPageGetDictionary

Returns the dictionary of a PDF page.

```
CGPDFDictionaryRef CGPDFPageGetDictionary (
   CGPDFPageRef page
);
```

Parameters

page

A PDF page.

Return Value
Returns the PDF dictionary for the specified page.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPDFPage.h`

CGPDFPageGetDocument

Returns the document for a page.

```
CGPDFDocumentRef CGPDFPageGetDocument (
   CGPDFPageRef page
);
```

Parameters

page

A PDF page.

Return Value
The PDF document with which the specified page is associated.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPDFPage.h`

CGPDFPageGetDrawingTransform

Returns the affine transform that maps a box to a given rectangle on a PDF page.

```
CGAffineTransform CGPDFPageGetDrawingTransform (
   CGPDFPageRef page,
   CGPDFBox box,
   CGRect rect,
   int rotate,
   bool preserveAspectRatio
);
```

Parameters

page

A PDF page.

box

A `CGPDFBox` constant that specifies the type of box. For possible values, see "PDF Boxes" (page 359).

rect

A Quartz rectangle.

rotate

An integer, that must be a multiple of `90`, that specifies the angle by which the specified rectangle is rotated clockwise.

preserveAspectRatio

A Boolean value that specifies whether or not the aspect ratio should be preserved. A value of `true` specifies that the aspect ratio should be preserved.

Return Value
An affine transform that maps the box specified by the `box` parameter to the rectangle specified by the `rect` parameter.

Discussion
Quartz constructs the affine transform as follows:

- Computes the effective rectangle by intersecting the rectangle associated with `box` and the `/MediaBox` entry of the specified page.
- Rotates the effective rectangle according to the page's `/Rotate` entry.
- Centers the resulting rectangle on `rect`. If the value of the `rotate` parameter is non-zero, then the rectangle is rotated clockwise by rotate degrees. The value of `rotate` must be a multiple of 90.
- Scales the rectangle, if necessary, so that it coincides with the edges of `rect`. If the value of `preserveAspectRatio` parameter is `true`, then the final rectangle coincides with the edges of `rect` only in the more restrictive dimension.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPDFPage.h`

CGPDFPageGetPageNumber

Returns the page number of the specified PDF page.

```
size_t CGPDFPageGetPageNumber (
   CGPDFPageRef page
);
```

Parameters

page

A PDF page.

Return Value
Returns the page number of the specified page.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGPDFPage.h

CGPDFPageGetRotationAngle

Returns the rotation angle of a PDF page.

```
int CGPDFPageGetRotationAngle (
   CGPDFPageRef page
);
```

Parameters

page
A PDF page.

Return Value
The rotation angle (in degrees) of the specified page. This is the value of the `/Rotate` entry in the page's dictionary.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGPDFPage.h

CGPDFPageGetTypeID

Returns the CFType ID for PDF page objects.

```
CFTypeID CGPDFPageGetTypeID (
   void
);
```

Return Value
Returns the Core Foundation type for a PDF page.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGPDFPage.h

CGPDFPageRelease

Decrements the retain count of a PDF page.

```
void CGPDFPageRelease (
   CGPDFPageRef page
);
```

Parameters

page
A PDF page.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `page` parameter is `NULL`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPDFPage.h`

CGPDFPageRetain

Increments the retain count of a PDF page.

```
CGPDFPageRef CGPDFPageRetain (
   CGPDFPageRef page
);
```

Parameters

page

A PDF page.

Return Value
The same page you passed in as the `page` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `page` parameter is `NULL`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPDFPage.h`

Data Types

CGPDFPageRef

An opaque type that represents a page in a PDF document.

```
typedef struct CGPDFPage *CGPDFPageRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPDFPage.h`

Constants

PDF Boxes

Box types for a PDF page.

```
enum CGPDFBox {
    kCGPDFMediaBox = 0,
    kCGPDFCropBox = 1,
    kCGPDFBleedBox = 2,
    kCGPDFTrimBox = 3,
    kCGPDFArtBox = 4
};
typedef enum CGPDFBox CGPDFBox;
```

Constants

`kCGPDFMediaBox`

The page media box—a rectangle, expressed in default user space units, that defines the boundaries of the physical medium on which the page is intended to be displayed or printed

Available in Mac OS X v10.3 and later.

Declared in `CGPDFPage.h`.

`kCGPDFCropBox`

The page crop box—a rectangle, expressed in default user space units, that defines the visible region of default user space. When the page is displayed or printed, its contents are to be clipped to this rectangle.

Available in Mac OS X v10.3 and later.

Declared in `CGPDFPage.h`.

`kCGPDFBleedBox`

The page bleed box—a rectangle, expressed in default user space units, that defines the region to which the contents of the page should be clipped when output in a production environment

Available in Mac OS X v10.3 and later.

Declared in `CGPDFPage.h`.

`kCGPDFTrimBox`

The page trim box—a rectangle, expressed in default user space units, that defines the intended dimensions of the finished page after trimming.

Available in Mac OS X v10.3 and later.

Declared in `CGPDFPage.h`.

`kCGPDFArtBox`

The page art box—a rectangle, expressed in default user space units, defining the extent of the page's meaningful content (including potential white space) as intended by the page's creator.

Available in Mac OS X v10.3 and later.

Declared in `CGPDFPage.h`.

Declared In

`CGPDFPage.h`

CGPDFScanner Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFScanner.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFScannerRef` opaque type is used to parse a PDF content stream. You can set up the PDF scanner object to invoke callbacks when it encounters specific PDF operators in the stream.

This opaque type is not derived from CFType and therefore there are no functions for retaining and releasing it.

Functions by Task

Creating a PDF Scanner Object

`CGPDFScannerCreate` (page 362)
Creates a CGPDFScanner object.

Retaining and Releasing PDF Scanner Objects

`CGPDFScannerRetain` (page 368)
Increments the retain count of a scanner object.

`CGPDFScannerRelease` (page 367)
Decrements the retain count of a scanner object.

Parsing Content

`CGPDFScannerScan` (page 368)
Parses the content stream of a CGPDFScanner object.

`CGPDFScannerGetContentStream` (page 363)
Returns the content stream associated with a CGPDFScanner object.

Getting PDF Objects from the Scanner Stack

`CGPDFScannerPopObject` (page 366)
Retrieves an object from the scanner stack.

`CGPDFScannerPopBoolean` (page 364)
Retrieves a Boolean object from the scanner stack.

`CGPDFScannerPopInteger` (page 364)
Retrieves an integer object from the scanner stack.

`CGPDFScannerPopNumber` (page 365)
Retrieves a real value object from the scanner stack.

`CGPDFScannerPopName` (page 365)
Retrieves a character string from the scanner stack.

`CGPDFScannerPopString` (page 367)
Retrieves a string object from the scanner stack.

`CGPDFScannerPopArray` (page 363)
Retrieves an array object from the scanner stack.

`CGPDFScannerPopDictionary` (page 364)
Retrieves a PDF dictionary object from the scanner stack.

`CGPDFScannerPopStream` (page 366)
Retrieves a PDF stream object from the scanner stack.

Functions

CGPDFScannerCreate

Creates a CGPDFScanner object.

```
CGPDFScannerRef CGPDFScannerCreate (
   CGPDFContentStreamRef cs,
   CGPDFOperatorTableRef table,
   void *info
);
```

Parameters

cs
A CGPDFContentStream object. (See *CGPDFContentStream Reference*.)

table
A CGPDFOperatorTable object that contains callbacks for the PDF operators you want to handle.

info
A pointer to data you want passed to your CGPDFOperatorTable callback function. (See *CGPDFOperatorTable Reference*.)

Return Value
A CGPDFScanner object. You are responsible for releasing this object by calling the function `CGPDFScannerRelease`.

Discussion
When you want to parse the contents of the PDF stream, call the function `CGPDFScannerScan` (page 368).

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFScanner.h`

CGPDFScannerGetContentStream

Returns the content stream associated with a CGPDFScanner object.

```
CGPDFContentStreamRef CGPDFScannerGetContentStream (
   CGPDFScannerRef scanner
);
```

Parameters

scanner
The scanner object whose content stream you want to obtain.

Return Value
Return the content stream associated with `scanner`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFScanner.h`

CGPDFScannerPopArray

Retrieves an array object from the scanner stack.

```
bool CGPDFScannerPopArray (
   CGPDFScannerRef scanner,
   CGPDFArrayRef *value
);
```

Parameters

scanner
A valid scanner object.

value
On output, points to the CGPDFArray object popped from the `scanner` stack.

Return Value
Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFScanner.h`

CGPDFScannerPopBoolean

Retrieves a Boolean object from the scanner stack.

```
bool CGPDFScannerPopBoolean (
   CGPDFScannerRef scanner,
   CGPDFBoolean *value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the CGPDFBoolean object popped from the `scanner` stack.

Return Value

Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFScanner.h`

CGPDFScannerPopDictionary

Retrieves a PDF dictionary object from the scanner stack.

```
bool CGPDFScannerPopDictionary (
   CGPDFScannerRef scanner,
   CGPDFDictionaryRef *value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the CGPDFDictionary object popped from the `scanner` stack.

Return Value

Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFScanner.h`

CGPDFScannerPopInteger

Retrieves an integer object from the scanner stack.

```
bool CGPDFScannerPopInteger (
   CGPDFScannerRef scanner,
   CGPDFInteger *value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the CGPDFInteger object popped from the `scanner` stack.

Return Value

Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFScanner.h`

CGPDFScannerPopName

Retrieves a character string from the scanner stack.

```
bool CGPDFScannerPopName (
   CGPDFScannerRef scanner,
   const char **value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the character string popped from the `scanner` stack.

Return Value

Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFScanner.h`

CGPDFScannerPopNumber

Retrieves a real value object from the scanner stack.

```
bool CGPDFScannerPopNumber (
   CGPDFScannerRef scanner,
   CGPDFReal *value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the CGPDFReal object popped from the `scanner` stack.

Return Value
Returns `true` if `value` is retrieved successfully; `false` otherwise.

Discussion
The number retrieved from the scanner can be a real value or an integer value. However, the result is always converted to a `CGPDFReal` data type.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFScanner.h`

CGPDFScannerPopObject

Retrieves an object from the scanner stack.

```
bool CGPDFScannerPopObject (
   CGPDFScannerRef scanner,
   CGPDFObjectRef *value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the object popped from the `scanner` stack.

Return Value
Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFScanner.h`

CGPDFScannerPopStream

Retrieves a PDF stream object from the scanner stack.

```
bool CGPDFScannerPopStream (
   CGPDFScannerRef scanner,
   CGPDFStreamRef *value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the CGPDFStream object popped from the `scanner` stack.

Return Value

Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFScanner.h`

CGPDFScannerPopString

Retrieves a string object from the scanner stack.

```
bool CGPDFScannerPopString (
   CGPDFScannerRef scanner,
   CGPDFStringRef *value
);
```

Parameters

scanner

A valid scanner object.

value

On output, points to the CGPDFString object popped from the `scanner` stack.

Return Value

Returns `true` if `value` is retrieved successfully; `false` otherwise.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFScanner.h`

CGPDFScannerRelease

Decrements the retain count of a scanner object.

```
void CGPDFScannerRelease (
   CGPDFScannerRef scanner
);
```

Parameters

scanner

The scanner object to release.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGPDFScanner.h

CGPDFScannerRetain

Increments the retain count of a scanner object.

```
CGPDFScannerRef CGPDFScannerRetain (
   CGPDFScannerRef scanner
);
```

Parameters

scanner

The scanner object to retain.

Return Value
The same scanner object passed to the function in the `scanner` parameter.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CGPDFScanner.h

CGPDFScannerScan

Parses the content stream of a CGPDFScanner object.

```
bool CGPDFScannerScan (
   CGPDFScannerRef scanner
);
```

Parameters

scanner

The scanner object whose content stream you want to parse.

Return Value
Returns `true` if the entire stream is parsed successfully; `false` if parsing fails (for example, if the stream data is corrupted).

Discussion
The function `CGPDFScannerScan` parses the PDF content stream associated with the scanner. Each time Quartz parses a PDF operator for which you register a callback, Quartz invokes your callback.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CGPDFScanner.h`

Data Types

CGPDFScannerRef

An opaque type used to parse a PDF content stream.

```
typedef struct CGPDFScanner *CGPDFScannerRef;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGPDFScanner.h`

CGPDFStream Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFStream.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFStreamRef` opaque type represents a PDF stream. A PDF stream consists of a dictionary that describes a sequence of bytes. Streams typically represent objects with potentially large amounts of data, such as images and page descriptions.

This opaque type is not derived from CFType and therefore there are no functions for retaining and releasing it.

Functions

CGPDFStreamCopyData

Returns the data associated with a PDF stream.

```
CFDataRef CGPDFStreamCopyData (
   CGPDFStreamRef stream,
   CGPDFDataFormat *format
);
```

Parameters

stream

A PDF stream.

format

On return, contains a constant that specifies the format of the data returned—`CGPDFDataFormatRaw` (page 373), `CGPDFDataFormatJPEGEncoded` (page 373), or `CGPDFDataFormatJPEG2000` (page 373).

Return Value

A CFData object that contains a copy of the stream data. You are responsible for releasing this object.

Availability

Available in Mac OS X version 10.3 and later.

Declared In
CGPDFStream.h

CGPDFStreamGetDictionary

Returns the dictionary associated with a PDF stream.

```
CGPDFDictionaryRef CGPDFStreamGetDictionary (
   CGPDFStreamRef stream
);
```

Parameters

stream
A PDF stream.

Return Value
The PDF dictionary for the specified stream.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CGPDFStream.h

Data Types

CGPDFStream

An opaque type that represents a PDF stream.

```
typedef struct CGPDFStream *CGPDFStreamRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGPDFStream.h

Constants

CGPDFDataFormat

The encoding format of PDF data.

```
enum CGPDFDataFormat {
 CGPDFDataFormatRaw,
 CGPDFDataFormatJPEGEncoded,
 CGPDFDataFormatJPEG2000
};
typedef enum CGPDFDataFormat CGPDFDataFormat;
```

Constants

`CGPDFDataFormatRaw`

The data stream is not encoded.

Available in Mac OS X v10.3 and later.

Declared in `CGPDFStream.h`.

`CGPDFDataFormatJPEGEncoded`

The data stream is encoded in JPEG format.

Available in Mac OS X v10.3 and later.

Declared in `CGPDFStream.h`.

`CGPDFDataFormatJPEG2000`

The data stream is encoded in JPEG-2000 format.

Available in Mac OS X v10.4 and later.

Declared in `CGPDFStream.h`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFStream.h`

CGPDFString Reference

Derived From:	None
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPDFString.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGPDFStringRef` opaque type represents a string in a PDF document A PDF string object of a series of bytes—unsigned integer values in the range 0 to 255. The string elements are not integer objects, but are stored in a more compact format. For more information on the representation of strings in PDF, see the latest version of *PDF Reference*, Adobe Systems Incorporated.

This opaque type is not derived from CFType and therefore there are no functions for retaining and releasing it. CGPDFString objects exist as constituent parts of a CGPDFDocument object, and are managed by their container.

Functions by Task

Converting PDF Strings

`CGPDFStringCopyTextString` (page 376)
Returns a CFString object that represents a PDF string as a text string.

`CGPDFStringCopyDate` (page 376)
Converts a string to a date.

Getting PDF String Data

`CGPDFStringGetBytePtr` (page 376)
Returns a pointer to the bytes of a PDF string.

`CGPDFStringGetLength` (page 377)
Returns the number of bytes in a PDF string.

Functions

CGPDFStringCopyDate

Converts a string to a date.

```
CFDateRef CGPDFStringCopyDate (
   CGPDFStringRef string
);
```

Parameters

string

The string to convert to a date.

Return Value

A CFDate object.

Discussion

The PDF specification defines a specific format for strings that represent dates. This function converts strings in that form to CFDate objects.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CGPDFString.h`

CGPDFStringCopyTextString

Returns a CFString object that represents a PDF string as a text string.

```
CFStringRef CGPDFStringCopyTextString (
   CGPDFStringRef string
);
```

Parameters

string

A PDF string. If this value is `NULL`, it will cause an error.

Return Value

Returns a CFString object that represents the specified PDF string as a text string. You are responsible for releasing this object.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFString.h`

CGPDFStringGetBytePtr

Returns a pointer to the bytes of a PDF string.

```
const unsigned char * CGPDFStringGetBytePtr (
   CGPDFStringRef string
);
```

Parameters

string

A PDF string.

Return Value

Returns a pointer to the bytes of the specified string. If the string is `NULL`, the function returns `NULL`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFString.h`

CGPDFStringGetLength

Returns the number of bytes in a PDF string.

```
size_t CGPDFStringGetLength (
   CGPDFStringRef string
);
```

Parameters

string

A PDF string.

Return Value

Returns the number of bytes referenced by the string, or `0` if the string is `NULL`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPDFString.h`

Data Types

CGPDFStringRef

An opaque data type that represents a string in a PDF document.

```
typedef struct CGPDFString *CGPDFStringRef;
```

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPDFString.h`

CGPSConverter Reference

Derived From:	*CFType Reference*
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGPSConverter.h
Companion guide	Quartz 2D Programming Guide

Overview

`CGPSConverterRef` is an opaque type used to convert PostScript data to PDF data. The PostScript data is supplied by a data provider and written into a data consumer. When you create a PostScript converter object, you can supply callback functions for Quartz to invoke at various stages of the conversion process,

Functions

CGPSConverterAbort

Tells a PostScript converter to abort a conversion at the next available opportunity.

```
bool CGPSConverterAbort (
   CGPSConverterRef converter
);
```

Parameters

converter

A PostScript converter.

Return Value

A Boolean value that indicates whether the converter is currently converting data (`true` if it is).

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPSConverter.h`

CGPSConverterConvert

Uses a PostScript converter to convert PostScript data to PDF data.

```
bool CGPSConverterConvert (
   CGPSConverterRef converter,
   CGDataProviderRef provider,
   CGDataConsumerRef consumer,
   CFDictionaryRef options
);
```

Parameters

converter

A PostScript converter.

provider

A Quartz data provider that supplies PostScript data.

consumer

A Quartz data provider that will receive the resulting PDF data.

options

This parameter should be `NULL`; it is reserved for future expansion of the API.

Return Value

A Boolean value that indicates whether the PostScript conversion completed successfully (`true` if it did).

Discussion

The conversion is thread safe, however it is not possible to have more than one conversion job in process within a given address space or process. If a given thread is running a conversion and another thread starts a new conversion, the second conversion will block until the first conversion is complete.

Important: Although `CGPSConverterConvert` is thread safe (it uses locks to prevent more than one conversion at a time in the same process), it is not thread safe with respect to the Resource Manager. If your application uses the Resource Manager on a separate thread, you should either use locks to prevent `CGPSConverterConvert` from executing during your usage of the Resource Manager or you should perform your conversions using the Post Script converter in a separate process.

In general, you can avoid this issue by using nib files instead of Resource Manager resources.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CGPSConverter.h`

CGPSConverterCreate

Creates a new PostScript converter.

```
CGPSConverterRef CGPSConverterCreate (
   void *info,
   const CGPSConverterCallbacks *callbacks,
   CFDictionaryRef options
);
```

Parameters

info

A pointer to the data that will be passed to the callbacks.

callbacks
A pointer to a PostScript converter callbacks structure that specifies the callbacks to be used during a conversion process.

options
This parameter should be `NULL`; it is reserved for future expansion of the API.

Return Value
A new PostScript converter, or `NULL` if a converter could not be created. You are responsible for releasing this object.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPSConverter.h`

CGPSConverterGetTypeID

Returns the Core Foundation type identifier for PostScript converters.

```
CFTypeID CGPSConverterGetTypeID (
   void
);
```

Return Value
The Core Foundation identifier for the opaque type `CGPSConverterRef` (page 386).

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPSConverter.h`

CGPSConverterIsConverting

Checks whether the converter is currently converting data.

```
bool CGPSConverterIsConverting (
   CGPSConverterRef converter
);
```

Parameters
converter
A PostScript converter.

Return Value
Returns `true` that indicates if the conversion is in progress.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CGPSConverter.h`

Callbacks by Task

Performing Custom Tasks at the Document Level

`CGPSConverterBeginDocumentCallback` (page 382)
Performs custom tasks at the beginning of a PostScript conversion process.

`CGPSConverterEndDocumentCallback` (page 383)
Performs custom tasks at the end of a PostScript conversion process.

Performing Custom Tasks at the Page Level

`CGPSConverterBeginPageCallback` (page 383)
Performs custom tasks at the beginning of each page in a PostScript conversion process.

`CGPSConverterEndPageCallback` (page 384)
Performs custom tasks at the end of each page of a PostScript conversion process.

Reporting Progress and Messages

`CGPSConverterProgressCallback` (page 385)
Reports progress periodically during a PostScript conversion process.

`CGPSConverterMessageCallback` (page 384)
Passes messages generated during a PostScript conversion process.

Performing Custom Clean-up Tasks

`CGPSConverterReleaseInfoCallback` (page 386)
Performs custom tasks when a PostScript converter is released.

Callbacks

CGPSConverterBeginDocumentCallback

Performs custom tasks at the beginning of a PostScript conversion process.

```
typedef void (*CGPSConverterBeginDocumentCallback)(void
*info);
```

If you name your function `MyConverterBeginDocument`, you would declare it like this:

```
size_t MyConverterBeginDocument (
    void *info
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPSConverterCreate` (page 380).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPSConverter.h`

CGPSConverterBeginPageCallback

Performs custom tasks at the beginning of each page in a PostScript conversion process.

```
typedef void (*CGPSConverterBeginPageCallback)(void
*info, size_t pageNumber, CFDictionaryRef pageInfo);
```

If you name your function `MyConverterBeginDocument`, you would declare it like this:

```
void MyConverterBeginPage (
    void *info,
   size_t pageNumber,
   CFDictionaryRef pageInfo
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPSConverterCreate` (page 380).

pageNumber

The current page number. Page numbers start at `1`.

pageInfo

A dictionary that contains contextual information about the page. This parameter is reserved for future API expansion, and is currently unused.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPSConverter.h`

CGPSConverterEndDocumentCallback

Performs custom tasks at the end of a PostScript conversion process.

```
typedef void (*CGPSConverterEndDocumentCallback)(void
*info, bool success);
```

If you name your function `MyConverterEndDocument`, you would declare it like this:

```
void MyConverterEndDocument (
    void *info,
```

```
    bool success
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPSConverterCreate` (page 380).

success

A Boolean value that indicates whether the PostScript conversion completed successfully (`true` if it did).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPSConverter.h`

CGPSConverterEndPageCallback

Performs custom tasks at the end of each page of a PostScript conversion process.

```
typedef void (*CGPSConverterEndPageCallback)(void
*info, size_t pageNumber, CFDictionaryRef pageInfo);
```

If you name your function `MyConverterEndPage`, you would declare it like this:

```
void MyConverterEndPage (
   void *info,
   size_t *pageNumber,
   CFDictionaryRef pageInfo
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPSConverterCreate` (page 380).

pageNumber

The current page number. Page numbers start at `1`.

pageInfo

A dictionary that contains contextual information about the page. This parameter is reserved for future API expansion, and is currently unused.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGPSConverter.h`

CGPSConverterMessageCallback

Passes messages generated during a PostScript conversion process.

```
typedef void (*CGPSConverterMessageCallback)(void
*info, CFStringRef message);
```

If you name your function `MyConverterMessage`, you would declare it like this:

```
void MyConverterMessage (
    void *info,
   CFStringRef message
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPSConverterCreate` (page 380).

message

A string containing the message from the PostScript conversion process.

Discussion

There are several kinds of message that might be sent during a conversion process. The most likely are font substitution messages, and any messages that the PostScript code itself generates. Any PostScript messages written to `stdout` are routed through this callback—typically these are debugging or status messages and, although uncommon, can be useful in debugging. In addition, there may be error messages if the document is malformed.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPSConverter.h`

CGPSConverterProgressCallback

Reports progress periodically during a PostScript conversion process.

```
typedef void (*CGPSConverterProgressCallback)(void
*info);
```

If you name your function `MyConverterProgress`, you would declare it like this:

```
void MyConverterProgress (
    void *info
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPSConverterCreate` (page 380).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPSConverter.h`

CGPSConverterReleaseInfoCallback

Performs custom tasks when a PostScript converter is released.

```
typedef void (*CGPSConverterReleaseInfoCallback)(void
*info);
```

If you name your function `MyConverterReleaseInfo`, you would declare it like this:

```
void MyConverterReleaseInfo (
  void *info
);
```

Parameters

info

A generic pointer to private data shared among your callback functions. This is the same pointer you supplied to `CGPSConverterCreate` (page 380).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPSConverter.h`

Data Types

CGPSConverterRef

An opaque data type used to convert PostScript data to PDF data.

```
typedef struct CGPSConverter *CGPSConverterRef;
```

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPSConverter.h`

CGPSConverterCallbacks

A structure for holding the callbacks provided when you create a PostScript converter object.

```
struct CGPSConverterCallbacks {
    unsigned int version;
    CGPSConverterBeginDocumentCallback beginDocument;
    CGPSConverterEndDocumentCallback endDocument;
    CGPSConverterBeginPageCallback beginPage;
    CGPSConverterEndPageCallback endPage;
    CGPSConverterProgressCallback noteProgress;
    CGPSConverterMessageCallback noteMessage;
    CGPSConverterReleaseInfoCallback releaseInfo;
};
typedef struct CGPSConverterCallbacks CGPSConverterCallbacks;
```

Fields

`version`
: The version number of the structure passed in as a parameter to the converter creation functions. The structure defined below is version `0`.

`beginDocument`
: The callback called at the beginning of the conversion of the PostScript document, or `NULL`.

`endDocument`
: The callback called at the end of conversion of the PostScript document, or `NULL`.

`beginPage`
: The callback called at the start of the conversion of each page in the PostScript document, or `NULL`.

`endPage`
: The callback called at the end of the conversion of each page in the PostScript document, or `NULL`.

`noteProgress`
: The callback called periodically during the conversion to indicate that conversion is proceeding, or `NULL`.

`noteMessage`
: The callback called to pass any messages that might result during the conversion, or `NULL`.

`releaseInfo`
: The callback called when the converter is deallocated, or `NULL`.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGPSConverter.h`

CGShading Reference

Derived From:	CFType
Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGShading.h
Companion guide	Quartz 2D Programming Guide

Overview

`CGShadingRef` is an opaque type used to define linear (axial) and radial gradient fills whose color transitions are controlled by a function (`CGFunctionRef` (page 199)) that you provide. Shading means to fill using a smooth transition between colors across an area. To paint with a Quartz shading, you call `CGContextDrawShading` (page 90). This function fills the current clipping path using the specified color gradient, calling your parametric function repeatedly as it draws

An alternative to using a CGShading object is to use the `CGGradientRef` (page 207) opaque type. For applications that run in Mac OS X v10.5 and later, CGGradient objects are much simpler to use. (See *CGGradient Reference*.)

Functions by Task

Creating Shading Objects

`CGShadingCreateAxial` (page 390)
Creates a shading object to use for axial shading.

`CGShadingCreateRadial` (page 391)
Creates a shading object to use for radial shading.

Retaining and Releasing Shading Objects

`CGShadingRetain` (page 392)
Increments the retain count of a shading object.

`CGShadingRelease` (page 392)
Decrements the retain count of a shading object.

Getting the CFType ID

`CGShadingGetTypeID` (page 391)
Returns the Core Foundation type identifier for Quartz shading objects.

Functions

CGShadingCreateAxial

Creates a shading object to use for axial shading.

```
CGShadingRef CGShadingCreateAxial (
   CGColorSpaceRef colorspace,
   CGPoint start,
   CGPoint end,
   CGFunctionRef function,
   bool extendStart,
   bool extendEnd
);
```

Parameters

colorspace
The color space in which color values are expressed. Quartz retains this object; upon return, you may safely release it.

start
The starting point of the axis, in the shading's target coordinate space.

end
The ending point of the axis, in the shading's target coordinate space.

function
A CGFunction object created by the function `CGFunctionCreate`. This object refers to your function for creating an axial shading. Quartz retains this object; upon return, you may safely release it.

extendStart
A Boolean value that specifies whether to extend the shading beyond the starting point of the axis.

extendEnd
A Boolean value that specifies whether to extend the shading beyond the ending point of the axis.

Return Value
A new Quartz axial shading. You are responsible for releasing this object using `CGShadingRelease` (page 392).

Discussion
An axial shading is a color blend that varies along a linear axis between two endpoints and extends indefinitely perpendicular to that axis. When you are ready to draw the shading, call the function `CGContextDrawShading` (page 90).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGShading.h`

CGShadingCreateRadial

Creates a shading object to use for radial shading.

```
CGShadingRef CGShadingCreateRadial (
   CGColorSpaceRef colorspace,
   CGPoint start,
   CGFloat startRadius,
   CGPoint end,
   CGFloat endRadius,
   CGFunctionRef function,
   bool extendStart,
   bool extendEnd
);
```

Parameters

colorspace
The color space in which color values are expressed. Quartz retains this object; upon return, you may safely release it.

start
The center of the starting circle, in the shading's target coordinate space.

startRadius
The radius of the starting circle, in the shading's target coordinate space.

end
The center of the ending circle, in the shading's target coordinate space.

endRadius
The radius of the ending circle, in the shading's target coordinate space.

function
A CGFunction object created by the function `CGFunctionCreate`. This object refers to your function for creating a radial shading. Quartz retains this object; upon return, you may safely release it.

extendStart
A Boolean value that specifies whether to extend the shading beyond the starting circle.

extendEnd
A Boolean value that specifies whether to extend the shading beyond the ending circle.

Return Value

A new Quartz radial shading. You are responsible for releasing this object using `CGShadingRelease` (page 392).

Discussion

A radial shading is a color blend that varies between two circles. To draw the shading, call the function `CGContextDrawShading` (page 90).

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CGShading.h`

CGShadingGetTypeID

Returns the Core Foundation type identifier for Quartz shading objects.

```
CFTypeID CGShadingGetTypeID (
   void
);
```

Return Value
The Core Foundation identifier for the opaque type `CGShadingRef` (page 393).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGShading.h`

CGShadingRelease

Decrements the retain count of a shading object.

```
void CGShadingRelease (
   CGShadingRef shading
);
```

Parameters

shading
The shading object to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the *shading* parameter is `NULL`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGShading.h`

CGShadingRetain

Increments the retain count of a shading object.

```
CGShadingRef CGShadingRetain (
   CGShadingRef shading
);
```

Parameters

shading
The shading object to retain.

Return Value
The same shading object you passed in as the `shading` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `shading` parameter is `NULL`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CGShading.h`

Data Types

CGShadingRef

An opaque type that represents a Quartz shading.

```
typedef struct CGShading *CGShadingRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGShading.h`

Managers

Apple Event Manager Reference

Framework:	CoreServices/CoreServices.h, Carbon/Carbon.h
Declared in	AEDataModel.h AEHelpers.h AEMach.h AEObjects.h AEPackObject.h AERegistry.h AEUserTermTypes.h AppleEvents.h

Overview

The Apple Event Manager, a part of the Open Scripting Architecture (OSA), provides facilities for applications to send and respond to Apple events and to make their operations and data available to AppleScript scripts. For related API reference, see Open Scripting Architecture Reference.

An Apple event is a type of interprocess message that can specify complex operations and data. Apple events provide a data transport and event dispatching mechanism that can be used within a single application, between applications on the same computer, and between applications on different computers connected to a network.

Applications typically use Apple events to request services and information from other applications or to provide services and information in response to such requests. All applications that present a graphical interface to the user through the Human Interface Toolbox (Carbon applications) or the Cocoa application framework should be able to respond, if appropriate, to certain events sent by the Mac OS. These include the `open application` (or `launch`), `reopen`, `open documents`, `print documents`, and `quit` events.

Some Apple Event Manager functions are marked as being thread safe—for all other functions, you should call them only on the main thread.

For an overview of technologies that take advantage of the Apple Event Manager, see AppleScript Overview.

For information on working with Apple events, including events sent by the Mac OS, see "Responding to Apple Events" in Apple Events Programming Guide. For information about individual four-character codes used in Apple events, see *AppleScript Terminology and Apple Event Codes Reference*.

The Apple Event Manager is implemented by the AE framework, a subframework of the Core Services framework. You don't link directly with the AE framework—instead, you typically link with the Carbon framework, which includes it. Some AppleEvent definitions are only available to clients of the Carbon framework, which includes, for example, `AEInteraction.h` in the HIToolbox framework.

The AE framework does not force a connection to the window server. This allows daemons and startup items that work with Apple events to continue working across log outs.

Functions by Task

Adding Items to Descriptor Lists

`AEPutArray` (page 459)
Inserts the data for an Apple event array into a descriptor list, replacing any previous descriptors in the list.

`AEPutDesc` (page 461)
Adds a descriptor to any descriptor list, possibly replacing an existing descriptor in the list.

`AEPutPtr` (page 465)
Inserts data specified in a buffer into a descriptor list as a descriptor, possibly replacing an existing descriptor in the list.

Adding Parameters and Attributes to Apple Events and Apple Event Records

`AEPutAttributeDesc` (page 460)
Adds a descriptor and a keyword to an Apple event as an attribute.

`AEPutAttributePtr` (page 461)
Adds a pointer to data, a descriptor type, and a keyword to an Apple event as an attribute.

`AEPutKeyDesc` (page 462)
Inserts a descriptor and a keyword into an Apple event record as an Apple event parameter.

`AEPutKeyPtr` (page 463)
Inserts data, a descriptor type, and a keyword into an Apple event record as an Apple event parameter.

`AEPutParamDesc` (page 464)
Inserts a descriptor and a keyword into an Apple event or Apple event record as an Apple event parameter.

`AEPutParamPtr` (page 464)
Inserts data, a descriptor type, and a keyword into an Apple event or Apple event record as an Apple event parameter.

Coercing Descriptor Types

`AECoerceDesc` (page 413)
Coerces the data in a descriptor to another descriptor type and creates a descriptor containing the newly coerced data.

`AECoercePtr` (page 414)
Coerces data to a desired descriptor type and creates a descriptor containing the newly coerced data.

Counting the Items in Descriptor Lists

`AECountItems` (page 415)
Counts the number of descriptors in a descriptor list.

Creating an Apple Event

`AECreateAppleEvent` (page 416)
Creates an Apple event with several important attributes but no parameters.

Creating and Duplicating Descriptors

`AECreateDesc` (page 417)
Creates a new descriptor that incorporates the specified data.

`AECreateDescFromExternalPtr` (page 418)
Creates a new descriptor that uses a memory buffer supplied by the caller.

`AEDuplicateDesc` (page 426)
Creates a copy of a descriptor.

Creating, Calling, and Deleting Universal Procedure Pointers

`DisposeAECoerceDescUPP` (page 503)
Disposes of a universal procedure pointer to a function that coerces data stored in a descriptor.

`DisposeAECoercePtrUPP` (page 503)
Disposes of a universal procedure pointer to a function that coerces data stored in a buffer.

`DisposeAEDisposeExternalUPP` (page 503)
Disposes of a universal procedure pointer to a function that disposes of data supplied to the `AECreateDescFromExternalPtr` function.

`DisposeAEEventHandlerUPP` (page 504)
Disposes of a universal procedure pointer to an event handler function.

`DisposeAEFilterUPP` (page 504)
Disposes of a universal procedure pointer to an Apple event filter function.

`DisposeAEIdleUPP` (page 504)
Disposes of a universal procedure pointer to an Apple event idle function.

`DisposeOSLAccessorUPP` (page 504)
Disposes of a universal procedure pointer to an object accessor function.

`DisposeOSLAdjustMarksUPP` (page 505)
Disposes of a universal procedure pointer to an object callback adjust marks function.

`DisposeOSLCompareUPP` (page 505)
Disposes of a universal procedure pointer to an object callback comparison function.

`DisposeOSLCountUPP` (page 505)
Disposes of a universal procedure pointer to an object callback count function.

`DisposeOSLDisposeTokenUPP` (page 506)
Disposes of a universal procedure pointer to an object callback dispose token function.

`DisposeOSLGetErrDescUPP` (page 506)
Disposes of a universal procedure pointer to an object callback get error descriptor function.

`DisposeOSLGetMarkTokenUPP` (page 506)
Disposes of a universal procedure pointer to an object callback get mark function.

DisposeOSLMarkUPP (page 507)
Disposes of a universal procedure pointer to an object callback mark function.

InvokeAECoerceDescUPP (page 507)
Calls a universal procedure pointer to a function that coerces data stored in a descriptor.

InvokeAECoercePtrUPP (page 508)
Calls a universal procedure pointer to a function that coerces data stored in a buffer.

InvokeAEDisposeExternalUPP (page 508)
Calls a dispose external universal procedure pointer.

InvokeAEEventHandlerUPP (page 509)
Calls an event handler universal procedure pointer.

InvokeAEFilterUPP (page 509)
Calls an Apple event filter universal procedure pointer.

InvokeAEIdleUPP (page 509)
Calls an Apple event idle universal procedure pointer.

InvokeOSLAccessorUPP (page 510)
Calls an object accessor universal procedure pointer.

InvokeOSLAdjustMarksUPP (page 510)
Calls an object callback adjust marks universal procedure pointer.

InvokeOSLCompareUPP (page 511)
Calls an object callback comparison universal procedure pointer.

InvokeOSLCountUPP (page 511)
Calls an object callback count universal procedure pointer.

InvokeOSLDisposeTokenUPP (page 512)
Calls an object callback dispose token universal procedure pointer.

InvokeOSLGetErrDescUPP (page 512)
Calls an object callback get error descriptor universal procedure pointer.

InvokeOSLGetMarkTokenUPP (page 513)
Calls an object callback get mark universal procedure pointer.

InvokeOSLMarkUPP (page 513)
Calls an object callback mark universal procedure pointer.

NewAECoerceDescUPP (page 514)
Creates a new universal procedure pointer to a function that coerces data stored in a descriptor.

NewAECoercePtrUPP (page 514)
Creates a new universal procedure pointer to a function that coerces data stored in a buffer.

NewAEDisposeExternalUPP (page 514)
Creates a new universal procedure pointer to a function that disposes of data stored in a buffer.

NewAEEventHandlerUPP (page 515)
Creates a new universal procedure pointer to an event handler function.

NewAEFilterUPP (page 515)
Creates a new universal procedure pointer to an Apple event filter function.

NewAEIdleUPP (page 515)
Creates a new universal procedure pointer to an Apple event idle function.

NewOSLAccessorUPP (page 516)
Creates a new universal procedure pointer to an object accessor function.

NewOSLAdjustMarksUPP (page 516)
Creates a new universal procedure pointer to an object callback adjust marks function.

NewOSLCompareUPP (page 517)
Creates a new universal procedure pointer to an object callback comparison function.

NewOSLCountUPP (page 517)
Creates a new universal procedure pointer to an object callback count function.

NewOSLDisposeTokenUPP (page 517)
Creates a new universal procedure pointer to an object callback dispose token function.

NewOSLGetErrDescUPP (page 518)
Creates a new universal procedure pointer to an object callback get error descriptor function.

NewOSLGetMarkTokenUPP (page 518)
Creates a new universal procedure pointer to an object callback get mark function.

NewOSLMarkUPP (page 518)
Creates a new universal procedure pointer to an object callback mark function.

Creating Descriptor Lists and Apple Event Records

AECreateList (page 419)
Creates an empty descriptor list or Apple event record.

Creating Object Specifiers

CreateCompDescriptor (page 498)
Creates a comparison descriptor that specifies how to compare one or more Apple event objects with either another Apple event object or a descriptor.

CreateLogicalDescriptor (page 499)
Creates a logical descriptor that specifies a logical operator and one or more logical terms for the Apple Event Manager to evaluate.

CreateObjSpecifier (page 500)
Assembles an object specifier that identifies one or more Apple event objects, from other descriptors.

CreateOffsetDescriptor (page 501)
Creates an offset descriptor that specifies the position of an element in relation to the beginning or end of its container.

CreateRangeDescriptor (page 502)
Creates a range descriptor that specifies a series of consecutive elements in the same container.

Deallocating Memory for Descriptors

AEDisposeDesc (page 424)
Deallocates the memory used by a descriptor.

Deallocating Memory for Tokens

`AEDisposeToken` (page 425)
Deallocates the memory used by a token.

Deleting Descriptors

`AEDeleteItem` (page 422)
Deletes a descriptor from a descriptor list, causing all subsequent descriptors to move up one place.

`AEDeleteKeyDesc` (page 423)
Deletes a keyword-specified parameter from an Apple event record.

`AEDeleteParam` (page 423)
Deletes a keyword-specified parameter from an Apple event record.

Dispatching Apple Events

`AEProcessAppleEvent` (page 457)
Calls the handler, if one exists, for a specified Apple event.

Getting, Calling, and Removing Object Accessor Functions

`AECallObjectAccessor` (page 412)
Invokes the appropriate object accessor function for a specific desired type and container type.

`AEGetObjectAccessor` (page 441)
Gets an object accessor function from an object accessor dispatch table.

`AEInstallObjectAccessor` (page 451)
Adds or replaces an entry for an object accessor function to an object accessor dispatch table.

`AERemoveObjectAccessor` (page 470)
Removes an object accessor function from an object accessor dispatch table.

Getting Data or Descriptors From Apple Events and Apple Event Records

`AEGetAttributeDesc` (page 429)
Gets a copy of the descriptor for a specified Apple event attribute from an Apple event; typically used when your application needs to pass the descriptor on to another function.

`AEGetAttributePtr` (page 430)
Gets a copy of the data for a specified Apple event attribute from an Apple event; typically used when your application needs to work with the data directly.

`AEGetKeyDesc` (page 436)
Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event record

`AEGetKeyPtr` (page 437)
Gets a copy of the data for a specified Apple event parameter from an Apple event record.

AEGetParamDesc (page 443)
Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event or an Apple event record.

AEGetParamPtr (page 444)
Gets a copy of the data for a specified Apple event parameter from an Apple event or an Apple event record.

Getting Information About the Apple Event Manager

AEManagerInfo (page 454)
Provides information about the version of the Apple Event Manager currently available or the number of processes that are currently recording Apple events.

Getting Items From Descriptor Lists

AEGetArray (page 428)
Extracts data from an Apple event array created with the AEPutArray function and stores it as a standard array of fixed size items in the specified buffer.

AEGetNthDesc (page 439)
Copies a descriptor from a specified position in a descriptor list into a specified descriptor; typically used when your application needs to pass the extracted data to another function as a descriptor.

AEGetNthPtr (page 440)
Gets a copy of the data from a descriptor at a specified position in a descriptor list; typically used when your application needs to work with the extracted data directly.

Getting the Sizes and Descriptor Types of Descriptors

AESizeOfAttribute (page 482)
Gets the size and descriptor type of an Apple event attribute from a descriptor of type AppleEvent.

AESizeOfKeyDesc (page 483)
Gets the size and descriptor type of an Apple event parameter from a descriptor of type AERecord.

AESizeOfNthItem (page 484)
Gets the data size and descriptor type of the descriptor at a specified position in a descriptor list.

AESizeOfParam (page 485)
Gets the size and descriptor type of an Apple event parameter from a descriptor of type AERecord or AppleEvent.

Initializing the Object Support Library

AEObjectInit (page 455)
Initializes the Object Support Library.

AESetObjectCallbacks (page 480)
Specifies the object callback functions for your application.

Locating Processes on Remote Computers

Available starting in Mac OS X version v10.3, these functions allow you to locate processes on remote computers (a task supported by the PPCToolbox in Mac OS 9).

`AECreateRemoteProcessResolver` (page 420)
Creates an object for resolving a list of remote processes.

`AEDisposeRemoteProcessResolver` (page 424)
Disposes of an `AERemoteProcessResolverRef`.

`AERemoteProcessResolverGetProcesses` (page 466)
Returns an array of objects containing information about processes running on a remote machine.

`AERemoteProcessResolverScheduleWithRunLoop` (page 467)
Schedules a resolver for execution on a given run loop in a given mode.

Managing Apple Event Dispatch Tables

`AEGetEventHandler` (page 435)
Gets an event handler from an Apple event dispatch table.

`AEInstallEventHandler` (page 449)
Adds an entry for an event handler to an Apple event dispatch table.

`AERemoveEventHandler` (page 469)
Removes an event handler entry from an Apple event dispatch table.

Managing Coercion Handler Dispatch Tables

`AEGetCoercionHandler` (page 431)
Gets the coercion handler for a specified descriptor type.

`AEInstallCoercionHandler` (page 448)
Installs a coercion handler in either the application or system coercion handler dispatch table.

`AERemoveCoercionHandler` (page 468)
Removes a coercion handler from a coercion handler dispatch table.

Managing Special Handler Dispatch Tables

`AEGetSpecialHandler` (page 446)
Gets a specified handler from a special handler dispatch table.

`AEInstallSpecialHandler` (page 452)
Installs a callback function in a special handler dispatch table.

`AERemoveSpecialHandler` (page 471)
Removes a handler from a special handler dispatch table.

Operating On Descriptor Data

`AEGetDescData` (page 432)
Gets the data from the specified descriptor.

`AEGetDescDataSize` (page 434)
Gets the size, in bytes, of the data in the specified descriptor.

`AEGetDescDataRange` (page 433)
Retrieves a specified series of bytes from the specified descriptor.

`AEReplaceDescData` (page 472)
Copies the specified data into the specified descriptor, replacing any previous data.

Requesting More Time to Respond to Apple Events

`AEResetTimer` (page 472)
Resets the timeout value for an Apple event to its starting value.

Requesting User Interaction

`AEGetInteractionAllowed` (page 436)
Gets your application's current user interaction preferences for responding to an Apple event as a server application.

`AEInteractWithUser` (page 453)
Initiates interaction with the user when your application is a server application responding to an Apple event.

`AESetInteractionAllowed` (page 479)
Specifies user interaction preferences for responding to an Apple event when your application is the server application.

Resolving Object Specifiers

`AEResolve` (page 473)
Resolves an object specifier.

Sending an Apple Event

`AESend` (page 476)
Sends the specified Apple event.

Creating Apple Event Structures in Memory

`AEBuildAppleEvent` (page 408)
Constructs an entire Apple event in a single call.

AEBuildDesc (page 410)
Provides a facility for compiling AEBuild descriptor strings into Apple event descriptors (AEDesc).

AEBuildParameters (page 411)
Adds additional parameters or attributes to an existing Apple event.

AEPrintDescToHandle (page 456)
Provides a pretty printer facility for displaying the contents of Apple event descriptors.

vAEBuildAppleEvent (page 519)
Allows you to encapsulate calls to AEBuildAppleEvent in a wrapper routine.

vAEBuildDesc (page 520)
Allows you to encapsulate calls to AEBuildDesc in your own wrapper routines.

vAEBuildParameters (page 521)
Allows you to encapsulate calls to AEBuildParameters in your own stdarg-style wrapper routines, using techniques similar to those allowed by vsprintf.

Creating Apple Event Structures Using Streams

AEStreamClose (page 485)
Closes and deallocates an AEStreamRef.

AEStreamCloseDesc (page 486)
Marks the end of a descriptor in an AEStreamRef.

AEStreamCloseList (page 486)
Marks the end of a list of descriptors in an AEStreamRef.

AEStreamCloseRecord (page 487)
Marks the end of a record in an AEStreamRef.

AEStreamCreateEvent (page 487)
Creates a new Apple event and opens a stream for writing data to it.

AEStreamOpen (page 489)
Opens a new AEStreamRef for use in building a descriptor.

AEStreamOpenDesc (page 489)
Marks the beginning of a descriptor in an AEStreamRef.

AEStreamOpenEvent (page 490)
Opens a stream for an existing Apple event.

AEStreamOpenKeyDesc (page 490)
Marks the beginning of a key descriptor in an AEStreamRef.

AEStreamOpenList (page 491)
Marks the beginning of a descriptor list in an AEStreamRef.

AEStreamOpenRecord (page 491)
Marks the beginning of an Apple event record in an AEStreamRef.

AEStreamOptionalParam (page 492)
Designates a parameter in an Apple event as optional.

AEStreamSetRecordType (page 493)
Sets the type of the most recently created record in an AEStreamRef.

AEStreamWriteAEDesc (page 493)
Copies an existing descriptor into an AEStreamRef.

AEStreamWriteData (page 494)
Appends data to the current descriptor in an AEStreamRef.

AEStreamWriteDesc (page 494)
Appends the data for a complete descriptor to an AEStreamRef.

AEStreamWriteKey (page 495)
Marks the beginning of a keyword/descriptor pair for a descriptor in an AEStreamRef.

AEStreamWriteKeyDesc (page 496)
Writes a complete keyword/descriptor pair to an AEStreamRef.

Working With Lower Level Apple Event Functions

AEGetRegisteredMachPort (page 445)
Returns the Mach port (in the form of a mach_port_t) that was registered with the bootstrap server for this process.

AEDecodeMessage (page 421)
Decodes a Mach message and converts it into an Apple event and its related reply.

AESendMessage (page 478)
Sends an AppleEvent to a target process without some of the overhead required by AESend.

AEProcessMessage (page 458)
Decodes and dispatches a low level Mach message event to an event handler, including packaging and returning the reply to the sender.

Serializing Apple Event Data

AESizeOfFlattenedDesc (page 483)
Returns the amount of buffer space needed to store the descriptor after flattening it.

AEFlattenDesc (page 426)
Flattens the specified descriptor and stores the data in the supplied buffer.

AEUnflattenDesc (page 498)
Unflattens the data in the passed buffer and creates a descriptor from it.

Suspending and Resuming Apple Event Handling

AEGetTheCurrentEvent (page 447)
Gets the Apple event that is currently being handled.

AEResumeTheCurrentEvent (page 474)
Informs the Apple Event Manager that your application wants to resume the handling of a previously suspended Apple event or that it has completed the handling of the Apple event.

AESetTheCurrentEvent (page 481)
Specifies a current Apple event to take the place of the one your application has suspended.

AESuspendTheCurrentEvent (page 497)
Suspends the processing of the Apple event that is currently being handled.

Miscellaneous

`AECheckIsRecord` (page 413)
Determines whether a descriptor is truly an `AERecord`.

`AEInitializeDesc` (page 448)
Initializes a new descriptor.

Functions

AEBuildAppleEvent

Constructs an entire Apple event in a single call.

```
OSStatus AEBuildAppleEvent (
   AEEventClass theClass,
   AEEventID theID,
   DescType addressType,
   const void *addressData,
   Size addressLength,
   SInt16 returnID,
   SInt32 transactionID,
   AppleEvent *result,
   AEBuildError *error,
   const char *paramsFmt,
   ...
);
```

Parameters

theClass
The event class for the resulting Apple event. See `AEEventClass` (page 555).

theID
The event id for the resulting Apple event. See `AEEventID` (page 556).

addressType
The address type for the addressing information described in the next two parameters: usually one of `typeApplSignature`, `typeProcessSerialNumber`, or `typeKernelProcessID`. See `DescType` (page 560).

addressData
A pointer to the address information.

addressLength
The number of bytes pointed to by the `addressData` parameter.

returnID
The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID.

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events.

result

A pointer to a descriptor where the resulting descriptor should be stored. See `AppleEvent` (page 559) for a description of the data type.

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 546) for a description of the data type.

paramsFmt

An AEBuild format string describing the AppleEvent record to be created. The format of these strings is described in Technical Note TN2106, AEBuild*, AEPrint*, and Friends. That technote also describes possible error return codes for syntax errors in the format string.

Return Value

A numeric result code indicating the success of the call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. You can use the *error* parameter to discover information about other errors. See "Apple Event Manager Result Codes" (page 636).

Discussion

IMPORTANT: Following the parameters described above, the `AEBuildAppleEvent` function takes a variable number of parameters as specified by the format string provided in the *paramsFmt* parameter.

This function and related "AEBuild" routines (including `AEBuildDesc` (page 410) and `AEBuildParameters` (page 411), and the variable-argument versions, `vAEBuildAppleEvent` (page 519), `vAEBuildDesc` (page 520), and `vAEBuildParameters` (page 521)) provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `AEBuildAppleEvent` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

In many ways, the AEBuild routines are very much like the standard C library's `printf` suite of routines. The syntax for the format string that you provide is very simple and allows for the substitution of data items into the Apple event descriptors being created.

The `AEBuildAppleEvent` function is similar to `AECreateAppleEvent` (page 416), but in addition to creating the Apple event, it also constructs the parameters for the event from the last three arguments. You can use `AEBuildAppleEvent` to build an entire Apple event, or `AEBuildParameters` (page 411) to add additional parameters to an existing Apple event.

The syntax of the formatting string for an entire Apple event (as passed to `AEBuildAppleEvent`) is almost identical to that used to represent the contents of an Apple event, without the curly braces. The event is defined as a sequence of name-value pairs, with optional parameters preceded with a tilde (~) character. The syntax is described in Technical Note TN2106, AEBuild*, AEPrint*, and Friends.

It is important to note that the identifier for the direct parameter in an Apple event, specified by the constant `keyDirectObject`, is four minus signs (`'----'`). The minus sign has special meaning in AEBuild strings, and it should always be enclosed in single quotes when it is used to identify the direct parameter for an Apple event in a descriptor string.

Version Notes
Prior to Mac OS X version 10.3, `AEBuildAppleEvent` would fail if you supplied a data parameter with size greater than 32767 bytes.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEBuildDesc

Provides a facility for compiling AEBuild descriptor strings into Apple event descriptors (`AEDesc`).

```
OSStatus AEBuildDesc (
   AEDesc *dst,
   AEBuildError *error,
   const char *src,
   ...
);
```

Parameters

dst
: A pointer to a descriptor where the resulting descriptor should be stored. See `AEDesc` (page 546).

error
: A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 546).

src
: An *`AEBuild`* format string describing the descriptor to be created.

Return Value
A numeric result code indicating the success of the call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. You can use the *`error`* parameter to discover information about other errors. See also "Apple Event Manager Result Codes" (page 636).

Discussion
This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `AEBuildDesc` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

For additional information on using the AEBuild routines, see the descriptions for `AEBuildAppleEvent` (page 408) and `AEBuildParameters` (page 411).

Version Notes
Prior to Mac OS X version 10.3, `AEBuildDesc` would fail if you supplied a data parameter with size greater than 32767 bytes.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEBuildParameters

Adds additional parameters or attributes to an existing Apple event.

```
OSStatus AEBuildParameters (
   AppleEvent *event,
   AEBuildError *error,
   const char *format,
   ...
);
```

Parameters

event
The Apple event to which you are adding parameters. See `AppleEvent` (page 559).

error
A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 546).

format
An *AEBuild* format string describing the parameters to be created.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This function can be called more than once to add any desired number of parameters or attributes to an existing Apple event. The Apple event should already have been created through either a call to `AECreateAppleEvent` (page 416) or `AEBuildAppleEvent` (page 408).

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `AEBuildDesc` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

For additional information on using the AEBuild routines, see the descriptions for `AEBuildAppleEvent` (page 408) and `AEBuildDesc` (page 410).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AECallObjectAccessor

Invokes the appropriate object accessor function for a specific desired type and container type.

```
OSErr AECallObjectAccessor (
   DescType desiredClass,
   const AEDesc *containerToken,
   DescType containerClass,
   DescType keyForm,
   const AEDesc *keyData,
   AEDesc *token
);
```

Parameters

desiredClass

The type of the Apple event object requested. Some possible values are defined in "Object Class ID Constants" (page 599). See `DescType` (page 560).

containerToken

A pointer to the token that identifies the container for the desired object. (Token is defined in `AEDisposeToken` (page 425).) See `AEDesc` (page 546).

containerClass

The object class of the container for the desired objects. See `DescType` (page 560).

keyForm

The key form that specifies how to find the object within the container. Key form constants are described in "Key Form and Descriptor Type Object Specifier Constants" (page 590). See `DescType` (page 560).

keyData

A pointer to the key data that identifies the object within the container. The type of this data is form-specific. That is, `formName` typically has key data of type `typeText`. See `AEDesc` (page 546).

token

A pointer to a token. On return, a token specifying the desired object (or objects). Your application should dispose of this token when it is through with it by calling `AEDisposeToken` (page 425). See `AEDesc` (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). `AECallObjectAccessor` returns any result codes returned by the object accessor function it calls.

Discussion

If you want your application to do some of the Apple event object resolution normally performed by the `AEResolve` (page 473) function, you can use `AECallObjectAccessor` to invoke an object accessor function. This might be useful, for example, if you have installed an object accessor function using `typeWildCard` for the `AEInstallObjectAccessor` function's `desiredClass` parameter and `typeAEList` for the `containerType` parameter. To return a list of tokens for a request like "line one of every window" the object accessor function can create an empty list, then call `AECallObjectAccessor` for each requested element, adding tokens for each element to the list one at a time.

The parameters of `AECallObjectAccessor` are identical to the parameters of an object accessor function, as described in `OSLAccessorProcPtr` (page 533) with one exception—the Apple Event Manager adds a reference constant parameter each time it calls the object accessor function.

You can also call a specific object accessor function directly through its universal procedure pointer with one of the invoke functions described in "Creating, Calling, and Deleting Universal Procedure Pointers" (page 399).

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AECheckIsRecord

Determines whether a descriptor is truly an `AERecord`.

```
Boolean AECheckIsRecord (
   const AEDesc *theDesc
);
```

Parameters

`theDesc`
A pointer to the descriptor to check.

Return Value
Returns `true` if the descriptor is an `AERecord` or an `AppleEvent`, `false` otherwise.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoerceDesc

Coerces the data in a descriptor to another descriptor type and creates a descriptor containing the newly coerced data.

```
OSErr AECoerceDesc (
   const AEDesc *theAEDesc,
   DescType toType,
   AEDesc *result
);
```

Parameters

`theAEDesc`
A pointer to the descriptor containing the data to coerce. See `AEDesc` (page 546).

toType

The desired descriptor type of the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

result

A pointer to a descriptor. On successful return, a descriptor containing the coerced data and matching the descriptor type specified in `toType`. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). If `AECoerceDesc` returns a nonzero result code, it returns a null descriptor record (a descriptor record of type `typeNull`, which does not contain any data) unless the Apple Event Manager is not available because of limited memory.

Version Notes

See the Version Notes section for the `AECoercePtr` (page 414) function for information on when to use descriptor-based versus pointer-based coercion handlers starting in Mac OS X version 10.2.

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`AEDataModel.h`

AECoercePtr

Coerces data to a desired descriptor type and creates a descriptor containing the newly coerced data.

```
OSErr AECoercePtr (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   DescType toType,
   AEDesc *result
);
```

Parameters

typeCode

The descriptor type of the source data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr

A pointer to the data to coerce.

dataSize

The length, in bytes, of the data to coerce.

toType

The desired descriptor type of the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

result

A pointer to a descriptor. On successful return, a descriptor containing the coerced data and matching the descriptor type specified in `toType`. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Starting in Mac OS X version 10.2, pointer-based coercion handlers are not called if the input type is "structured"—that is, if the type to be coerced is `typeAEList`, `typeAERecord`, or coerced `typeAERecord`. If you want to add a coercion handler for one of these types, it must be a descriptor-based handler. This does not mean you are required to use descriptor-based coercion handlers everywhere—for "flat" data types, such as `typeText`, pointer-based handlers are still fine.

Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECountItems

Counts the number of descriptors in a descriptor list.

```
OSErr AECountItems (
   const AEDescList *theAEDescList,
   long *theCount
);
```

Parameters

theAEDescList

A pointer to the descriptor list to count. See `AEDescList` (page 553).

theCount

A pointer to a count variable. On return, the number of descriptors in the specified descriptor list, which can be 0, if the list is empty.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Your application typically counts the descriptors in a descriptor list when it is extracting data from an Apple event. You can use the functions in "Getting Items From Descriptor Lists" to get an individual item from a descriptor list or to iterate through the items.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`AEDataModel.h`

AECreateAppleEvent

Creates an Apple event with several important attributes but no parameters.

```
OSErr AECreateAppleEvent (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   const AEAddressDesc *target,
   AEReturnID returnID,
   AETransactionID transactionID,
   AppleEvent *result
);
```

Parameters

theAEEventClass

The event class of the Apple event to create. This parameter becomes accessible through the `keyEventClassAttr` attribute of the Apple event. Some event classes are described in "Event Class Constants" (page 585). See `AEEventClass` (page 555).

theAEEventID

The event ID of the Apple event to create. This parameter becomes accessible through the `keyEventIDAttr` attribute of the Apple event. Some event IDs are described in "Event ID Constants" (page 586). See `AEEventID` (page 556).

target

A pointer to an address descriptor. Before calling `AECreateAppleEvent`, you set the descriptor to identify the target (or server) application for the Apple event. This parameter becomes accessible through the `keyAddressAttr` attribute of the Apple event. See `AEAddressDesc` (page 551).

returnID

The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID. This parameter becomes accessible through the `keyReturnIDAttr` attribute of the Apple event. The return ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 589). See `AEReturnID` (page 558).

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events. This parameter becomes accessible through the `keyTransactionIDAttr` attribute of the Apple event. This transaction ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 589). See `AETransactionID` (page 559).

result

A pointer to an Apple event. On successful return, the new Apple event. On error, a null descriptor (one with descriptor type `typeNull`). If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting Apple event after it has finished using it. See the `AppleEvent` (page 559) data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The `AECreateAppleEvent` function creates an empty Apple event. You can add parameters to the Apple event after you create it with the functions described in "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 398).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECreateDesc

Creates a new descriptor that incorporates the specified data.

```
OSErr AECreateDesc (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   AEDesc *result
);
```

Parameters

typeCode

The descriptor type for the new descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr

A pointer to the data for the new descriptor. This data is copied into a newly-allocated block of memory for the descriptor that is created. To minimize copying overhead, consider using `AECreateDescFromExternalPtr` (page 418).

dataSize

The length, in bytes, of the data for the new descriptor.

result

A pointer to a descriptor. On successful return, a descriptor that incorporates the data specified by the `dataPtr` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
While it is possible to create an Apple event descriptor or a descriptor list or a descriptor with the `AECreateDesc` function (assuming you have access to the raw data for an Apple event, list, or descriptor), you typically create these structured objects with their specific creation routines—`AECreateAppleEvent`, `AECreateList`, or `AECreateDesc`.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECreateDescFromExternalPtr

Creates a new descriptor that uses a memory buffer supplied by the caller.

```
OSStatus AECreateDescFromExternalPtr (
   OSType descriptorType,
   const void *dataPtr,
   Size dataLength,
   AEDisposeExternalUPP disposeCallback,
   SRefCon disposeRefcon,
   AEDesc *theDesc
);
```

Parameters

descriptorType
The descriptor type for the new descriptor.

dataPtr
A pointer to the data for the new descriptor. The memory that is pointed to cannot be a `Handle` (which may move in memory), cannot be modified by the caller, and must be preserved in place (and not freed), until the *disposeCallback* function is called.

If possible, the descriptor will be mapped into the address space of the recipient using shared memory, avoiding an actual memory copy.

The pointer that is passed in does not need to be aligned to any particular boundary, but is optimized to transfer data on a page boundary. You can get the current page size (4096 on all current Mac OS X systems) with the `getpagesize(3)` call. (Type `man 3 getpagesize` in a Terminal window for documentation.)

dataLength
The length, in bytes, of the data for the new descriptor.

disposeCallback
A universal procedure pointer to a dispose callback function of type `AEDisposeExternalProcPtr` (page 527). Your callback function will be called when the block of memory provided by `dataPtr` is no longer needed by the Apple Event Manager. The function can be called at any time, including during creation of the descriptor.

disposeRefcon

A reference constant the Apple Event Manager passes to the `disposeCallback` function whenever it calls the function. If your dispose function doesn't require a reference constant, pass 0 for this parameter.

theDesc

A pointer to a descriptor. On successful return, a descriptor that incorporates the data specified by the `dataPtr` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This function is different than `AECreateDesc` (page 417), in that it creates a descriptor that uses the data block provided by the caller "in place," rather than allocate a block of memory and copy the data to it. This function can provide dramatically improved performance if you're working with large chunks of data. It attempts to copy the descriptor to the address space of any recipient process using virtual memory APIs, avoiding an actual memory copy. For example, you might want to use this function to pass a large image in an Apple event.

You can use the `AEGetDescDataRange` (page 433) function to access a specific section of a large block of data.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

AECreateList

Creates an empty descriptor list or Apple event record.

```
OSErr AECreateList (
   const void *factoringPtr,
   Size factoredSize,
   Boolean isRecord,
   AEDescList *resultList
);
```

Parameters

factoringPtr

A pointer to the data at the beginning of each descriptor that is the same for all descriptors in the list. If there is no common data, or if you decide not to isolate the common data, pass `NULL` as the value of this parameter.

factoredSize

The size of the common data. If there is no common data, or if you decide not to isolate the common data, pass 0 as the value of this parameter. (See the Discussion section for more information.)

isRecord

A Boolean value that specifies the kind of list to create. Pass a value of `TRUE` to create an Apple event record (a data structure of type `AERecord` (page 557)) or `FALSE` to create a descriptor list.

resultList

A pointer to a descriptor list variable. On successful return, the descriptor list or Apple event record that the `AECreateList` function creates. On error, a null descriptor. See `AEDescList` (page 553).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The `AECreateList` function creates an empty descriptor list or Apple event record. You can use the functions described in "Adding Items to Descriptor Lists" to populate the list as part of creating an Apple event. After sending the Apple event with the `AESend` (page 476) function, you should dispose of the descriptor list with the `AEDisposeDesc` (page 424) function when you no longer need it.

If you intend to use a descriptor list for a factored Apple event array, you must provide, in the `factoringPtr` parameter, a pointer to the data shared by all items in the array and, in the `factoredSize` parameter, the size of the common data. The common data must be 4, 8, or more than 8 bytes in length because it always consists of (a) the descriptor type (4 bytes) (b) the descriptor type (4 bytes) and the size of each item's data (4 bytes) or (c) the descriptor type (4 bytes), the size of each item's data (4 bytes), and some portion of the data itself (1 or more bytes).

For information about data types used with Apple event arrays, see "Apple Event Manager Data Types" (page 545).

Version Notes
The *factoringPtr* and *factoredSize* parameters are not supported in Mac OS X v10.2 and later. You should pass `NULL` and zero, respectively, for these parameters.

Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECreateRemoteProcessResolver

Creates an object for resolving a list of remote processes.

```
AERemoteProcessResolverRef AECreateRemoteProcessResolver (
   CFAllocatorRef allocator,
   CFURLRef url
);
```

Parameters

allocator

An object that is used to allocates and deallocate any Core Foundation types created or returned by this API. You can pass `kCFAllocatorDefault` to get the default allocation behavior. The allocator is based on `CFAllocatorRef`, an opaque data type described in the Core Foundation Reference Documentation.

url

A `CFURL` reference identifying the remote host and port on which to look for processes. See the Core Foundation Reference Documentation for a description of the `CFURLRef` data type.

Return Value
An `AERemoteProcessResolverRef` (page 557), which must be disposed of with `AEDisposeRemoteProcessResolver` (page 424). A resolver can only be used one time; once it has obtained a list of remote processes from a server, or gotten an error, it can no longer be scheduled. To retrieve a new list of processes, create a new instance of this object.

Discussion
You supply this function with the URL for a remote host and port; it returns a reference to a resolver object. To obtain a list of remote processes from the resolver, you can query it synchronously with `AERemoteProcessResolverGetProcesses` (page 466), which blocks until the request completes (either successfully or with an error).

If asynchronous behavior is desired, you can optionally use `AERemoteProcessResolverScheduleWithRunLoop` (page 467) to schedule the resolver asynchronously on a run loop. If so, you supply a callback routine (see `AERemoteProcessResolverCallback` (page 532)) that is executed when the resolver completes. To obtain information about the remote processes, you will again have to call `AERemoteProcessResolverGetProcesses` (page 466).

A resolver can only be used once; once it has fetched the data or gotten an error it can no longer be scheduled. The data obtained by the resolver is a `CFArrayRef` of `CFDictionaryRef` objects. For information on the format of the returned remote process information, see the description of the function result for the function `AERemoteProcessResolverGetProcesses` (page 466), and also "Remote Process Dictionary Keys" (page 602).

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AEDecodeMessage

Decodes a Mach message and converts it into an Apple event and its related reply.

```
OSStatus AEDecodeMessage (
   mach_msg_header_t *header,
   AppleEvent *event,
   AppleEvent *reply
);
```

Parameters

header

A pointer to a Mach message header for the event to be decoded.

event

A pointer to a null Apple event descriptor (one with descriptor type `typeNull`). On successful completion, contains the decoded Apple event. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it.

reply

A pointer to a null Apple event descriptor. On successful completion, contains the reply event from the decoded Apple event. To send the reply, you use the following:

```
AESendMessage(reply, NULL, kAENoReply, kAEDefaultTimeout);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The Apple Event Manager provides the following functions (on Mac OS X only) for working with Apple events at a lower level: `AEGetRegisteredMachPort` (page 445), `AEDecodeMessage`, `AESendMessage` (page 478), and `AEProcessMessage` (page 458). See the descriptions for those functions for more information on when you might use them.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEMach.h`

AEDeleteItem

Deletes a descriptor from a descriptor list, causing all subsequent descriptors to move up one place.

```
OSErr AEDeleteItem (
   AEDescList *theAEDescList,
   long index
);
```

Parameters

theAEDescList

A pointer to the descriptor list containing the descriptor to delete. See `AEDescList` (page 553).

index

A one-based positive integer indicating the position of the descriptor to delete. `AEDeleteItem` returns an error if you pass zero, a negative number, or a value that is out of range.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDeleteKeyDesc

Deletes a keyword-specified parameter from an Apple event record.

```
OSErr AEDeleteKeyDesc (
    AERecord *theAERecord,
    AEKeyword theAEKeyword
);
```

Parameters

theAERecord

A pointer to the Apple event record to delete the parameter from.

theAEKeyword

The keyword that specifies the parameter to delete. Some keyword constants are described in "Keyword Parameter Constants" (page 595). See `AEKeyword` (page 556).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

This function is declared as a macro that invokes `AEDeleteParam` (page 423), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes

See `AEDeleteParam` (page 423).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AEDeleteParam

Deletes a keyword-specified parameter from an Apple event record.

```
OSErr AEDeleteParam (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword
);
```

Parameters

theAppleEvent

A pointer to the Apple event or Apple event record to delete the parameter from. See `AppleEvent` (page 559).

theAEKeyword

The keyword that specifies the parameter to delete. Some keyword constants are described in "Keyword Parameter Constants" (page 595). See `AEKeyword` (page 556).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEDisposeDesc

Deallocates the memory used by a descriptor.

```
OSErr AEDisposeDesc (
   AEDesc *theAEDesc
);
```

Parameters

theAEDesc

A pointer to the descriptor to deallocate. On return, a null descriptor. If you pass a null descriptor in this parameter, AEDisposeDesc returns noErr. See AEDesc (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). As currently implemented, AEDisposeDesc always returns noErr.

Discussion
The AEDisposeDesc function deallocates the memory used by a descriptor. After calling this method, the descriptor becomes an empty descriptor with a type of typeNULL. Because all Apple event structures (except for keyword-specified descriptors) are descriptors, you can use AEDisposeDesc for any of them.

Do not call AEDisposeDesc on a descriptor obtained from another Apple Event Manager function (such as the reply event from a call to AESend (page 476)) unless that function returns successfully.

Special Considerations

If the AEDesc might contain an OSL token, dispose of it with AEDisposeToken (page 425).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
AEDataModel.h

AEDisposeRemoteProcessResolver

Disposes of an AERemoteProcessResolverRef.

```
void AEDisposeRemoteProcessResolver (
   AERemoteProcessResolverRef ref
);
```

Parameters

ref

The `AERemoteProcessResolverRef` (page 557) to dispose of. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 420).

Discussion
If this resolver is currently scheduled on a run loop, it is unscheduled, and the asynchronous callback is not executed.

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AEDisposeToken

Deallocates the memory used by a token.

```
OSErr AEDisposeToken (
   AEDesc *theToken
);
```

Parameters

theToken

A pointer to the token to dispose of. On successful return, the pointer is set to the null descriptor. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Your application calls the `AEResolve` (page 473) function to resolve an object specifier, with the help of the object accessor functions described in "Object Accessor Callbacks" and the application object callback functions described in "Object Callback Functions".

When `AEResolve` returns a final token to your event handler as the result of the resolution of an object specifier, your application must deallocate the memory used by the token. When your application calls the `AEDisposeToken` function, the Apple Event Manager first calls your application's token disposal function, if you have provided one. The token disposal function is described in `OSLDisposeTokenProcPtr` (page 539).

If you haven't provided a token disposal function, or if your application's token disposal function returns `errAEEventNotHandled` as the function result, the Apple Event Manager calls the system token disposal function if one is available. If there is no system token disposal function or the function returns `errAEEventNotHandled` as the function result, the Apple Event Manager calls the `AEDisposeDesc` function to dispose of the token.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEDuplicateDesc

Creates a copy of a descriptor.

```
OSErr AEDuplicateDesc (
   const AEDesc *theAEDesc,
   AEDesc *result
);
```

Parameters

theAEDesc

A pointer to the descriptor to duplicate. See `AEDesc` (page 546).

result

A pointer to a descriptor. On return, the descriptor contains a copy of the descriptor specified by the `theAEDesc` parameter. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
It is common for applications to send Apple events that have one or more attributes or parameters in common. For example, if you send a series of Apple events to the same application, the address attribute is the same. In these cases, the most efficient way to create the necessary Apple events is to make a template Apple event that you can then copy—by calling the `AEDuplicateDesc` function—as needed. You then fill in or change the remaining parameters and attributes of the copy, send the copy by calling the `AESend` (page 476) function and, after `AESend` returns a result code, dispose of the copy by calling `AEDisposeDesc` (page 424). You can use this approach to prepare structures of type `AEDesc` (page 546), `AEDescList` (page 553), `AERecord` (page 557), and `AppleEvent` (page 559).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEFlattenDesc

Flattens the specified descriptor and stores the data in the supplied buffer.

```
OSStatus AEFlattenDesc (
   const AEDesc *theAEDesc,
   Ptr buffer,
   Size bufferSize,
   Size *actualSize
);
```

Parameters

theAEDesc

A pointer to the descriptor to be flattened. See `AEDesc` (page 546).

buffer

A pointer to memory, allocated by the application, where the flattened data will be stored. See the *bufferSize* parameter for information on how large a buffer you should allocate.

bufferSize

The size of the buffer pointed to by buffer. Prior to calling `AEFlattenDesc`, you call the `AESizeOfFlattenedDesc` (page 483) function to determine the required size of the buffer for the flatten operation.

If *bufferSize* is too small, `AEFlattenDesc` returns `errAEBufferTooSmall` and doesn't store any data in the buffer.

actualSize

A pointer to a size variable. On return, the variable contains the actual size of the flattened data. You can specify `NULL` for this parameter if you do not care about the returned size.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

Flattening a descriptor serializes the data it contains. That is, it reduces a complex, possibly deeply nested structure to a series of bytes that can conveniently be stored. The descriptor can be reconstituted from the stored bytes with the `AEUnflattenDesc` (page 498) function.

Applications can be scriptable and work with Apple events without needing to flatten and unflatten descriptors. Flattening is a special-purpose capability that is useful in circumstances where it may be convenient to store data by saving and restoring a descriptor, rather than having to manually extract the data from it, store the data as a separate step, then manually recreate the descriptor (if necessary). For example, you might use flattening to store a preference setting received through an Apple event.

Flattening and unflattening should work without loss of data on descriptors that represent `AEDesc`, `AEList`, and `AERecord` structures. You can also use the process with `AppleEvent` descriptors. However, keep in mind that Apple events may contain attributes that are relevant only to a running process, and these attributes may not keep their meaning when the event is reconstituted.

Flattening and unflattening works across OS versions, including between Mac OS 9 and Mac OS X.

Flattening is endian-neutral. That is, you can save flattened data on a machine that is either big-endian or little-endian, then retrieve and unflatten the data on either type of machine, without any special steps by your application.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEGetArray

Extracts data from an Apple event array created with the AEPutArray function and stores it as a standard array of fixed size items in the specified buffer.

```
OSErr AEGetArray (
   const AEDescList *theAEDescList,
   AEArrayType arrayType,
   AEArrayDataPointer arrayPtr,
   Size maximumSize,
   DescType *itemType,
   Size *itemSize,
   long *itemCount
);
```

Parameters

theAEDescList

A pointer to the descriptor list to get the array from. If the array is of type kAEDataArray, kAEPackedArray, or kAEHandleArray, the descriptor list must be factored. A factored descriptor list is one in which the Apple Event Manager automatically isolates the data that is common to all the elements of the list so that the common data only appears in the list once. To create a factored descriptor list, you call the AECreateList (page 419) function and specify the data that is common to all elements in the descriptor array. See the Discussion section for related information. See AEDescList (page 553).

arrayType

The Apple event array type to convert. Pass one of the constants: described in "Data Array Constants" (page 580). See AEArrayType (page 552).

arrayPtr

A pointer to a buffer, allocated and disposed of by your application, for storing the array. The size in bytes must be at least as large as the value you pass in the maximumSize parameter. On return, the buffer contains the array of fixed-size items. See AEArrayDataPointer (page 551).

maximumSize

The maximum length, in bytes, of the expected data. The AEGetArray function will not return more data than you specify in this parameter.

itemType

A pointer to a descriptor type. On return, for arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray, the descriptor type of the items in the returned array. The AEGetArray function doesn't supply a value in itemType for arrays of type kAEDescArray and kAEKeyDescArray because they may contain descriptors of different types. Possible descriptor types are listed in "Descriptor Type Constants" (page 581). See DescType (page 560).

itemSize

A pointer to a size variable. On return, for arrays of type kAEDataArray or kAEPackedArray, the size (in bytes) of each item in the returned array. You don't get an item size for arrays of type kAEDescArray, kAEKeyDescArray, or kAEHandleArray because descriptors and handles (though not the data they point to) have a known size.

itemCount

A pointer to a size variable. On return, the number of items in the returned array.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The `AEGetArray` function uses a buffer identified by the pointer in the `arrayPtr` parameter to store the converted data for the Apple event array specified by the `theAEDescList` parameter. For example, `AEGetArray` may convert an array of descriptors of type `typeLongInteger` into a simple array of integer values or an array of descriptors of type `typeFSS` into an array of file specification records.

Even if the descriptor list that contains the array is factored, the converted data for each array item includes the data common to all the descriptors in the list. The Apple Event Manager automatically reconstructs the common data for each item when you call `AEGetArray`.

For information about creating and factoring descriptor lists for Apple event arrays, see `AECreateList` (page 419). For information about adding an Apple event array to a descriptor list, see `AEPutArray` (page 459).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetAttributeDesc

Gets a copy of the descriptor for a specified Apple event attribute from an Apple event; typically used when your application needs to pass the descriptor on to another function.

```
OSErr AEGetAttributeDesc (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType desiredType,
   AEDesc *result
);
```

Parameters

theAppleEvent
A pointer to the Apple event to get the attribute descriptor from. See `AppleEvent` (page 559).

theAEKeyword
The keyword that specifies the desired attribute. Some keyword constants are described in "Keyword Attribute Constants" (page 593). See `AEKeyword` (page 556).

result
A pointer to a descriptor. On successful return, a copy of the specified Apple event attribute, coerced, if necessary, to the descriptor type specified in `desiredType`. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
To get Apple event attribute data for your application to use directly, call `AEGetAttributePtr` (page 430). To get a descriptor for an Apple event attribute to pass on to another Apple Event Manager routine, call `AEGetAttributeDesc`.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetAttributePtr

Gets a copy of the data for a specified Apple event attribute from an Apple event; typically used when your application needs to work with the data directly.

```
OSErr AEGetAttributePtr (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType desiredType,
   DescType *typeCode,
   void *dataPtr,
   Size maximumSize,
   Size *actualSize
);
```

Parameters

theAppleEvent

A pointer to the Apple event to get the attribute data from. See `AppleEvent` (page 559).

theAEKeyword

The keyword that specifies the desired attribute. Some keyword constants are described in "Keyword Attribute Constants" (page 593). See `AEKeyword` (page 556).

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If the descriptor specified by the `theAEKeyword` parameter is not of the desired type, `AEGetAttributePtr` attempts to coerce the data to this type. However, if you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the Apple event attribute.

On return, you can determine the actual descriptor type by examining the `typeCode` parameter.

See `DescType` (page 560).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the attribute data pointed to by `dataPtr`. The returned type is either the same as the type specified by the `desiredType` parameter or, if the desired type was type wildcard, the true type of the descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr

A pointer to a buffer, local variable, or other storage location, created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the attribute data.

maximumSize

The maximum length, in bytes, of the expected attribute data. The `AEGetAttributePtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a size variable. On return, the length, in bytes, of the data for the specified Apple event attribute. If this value is larger than the value you passed in the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the attribute, though `AEGetAttributePtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetAttributePtr` again.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
To get Apple event attribute data for your application to use directly, call `AEGetAttributePtr`. To get a descriptor for an Apple event attribute to pass on to another Apple Event Manager routine, call `AEGetAttributeDesc` (page 429).

Before calling `AEGetAttributePtr`, you can call the `AESizeOfAttribute` (page 482) function to determine a size for the `dataPtr` buffer. However, unless you specify `typeWildCard` for the `desiredType` parameter, `AEGetAttributePtr` may coerce the data, which may cause the size of the data to change.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetCoercionHandler

Gets the coercion handler for a specified descriptor type.

```
OSErr AEGetCoercionHandler (
   DescType fromType,
   DescType toType,
   AECoercionHandlerUPP *handler,
   SRefCon *handlerRefcon,
   Boolean *fromTypeIsDesc,
   Boolean isSysHandler
);
```

Parameters

fromType

The descriptor type of the data coerced by the handler. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

toType
The descriptor type of the resulting data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

handler
A universal procedure pointer. On return, a pointer to the specified handler, if a coercion table entry exists that exactly matches the values supplied in the parameters `fromType` and `toType`. See `AECoercionHandlerUPP` (page 552).

handlerRefcon
A pointer to a reference constant. On return, the reference constant from the coercion table entry for the specified coercion handler. The Apple Event Manager passes this reference constant to the handler each time it calls the handler. The reference constant may have a value of 0.

fromTypeIsDesc
A pointer to a Boolean value. The `AEGetCoercionHandler` function returns a value of `TRUE` in this parameter if the coercion handler expects the data as a descriptor or `FALSE`, if the coercion handler expects a pointer to the data.

isSysHandler
Specifies the coercion table to get the handler from. Pass `TRUE` to get the handler from the system coercion table or `FALSE` to get the handler from your application's coercion table. Use of the system coercion table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Your application should not install a coercion handler in a system coercion handler dispatch table with the goal that the handler will get called when other applications perform coercions—this won't work in Mac OS X. For more information, see "Writing and Installing Coercion Handlers" in Apple Events Programming Guide.

In Mac OS 7.1 through 9.x and Mac OS X version v10.2 and later, `AEGetCoercionHandler` returns `errAEHandlerNotInstalled` when there's not an exact match, even if a wildcard handler is installed that could handle the coercion. Mac OS X version v10.0.x and v10.1.x will return the wildcard handler.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetDescData

Gets the data from the specified descriptor.

```
OSErr AEGetDescData (
   const AEDesc *theAEDesc,
   void *dataPtr,
   Size maximumSize
);
```

Parameters

theAEDesc

A pointer to the descriptor to get the data from. See `AEDesc` (page 546).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes should be the same as the value you pass in the `maximumSize` parameter. On return, contains the data from the descriptor.

maximumSize

The length, in bytes, of the expected descriptor data. The `AEGetDescData` function will not return more data than you specify in this parameter. You typically determine the maximum size by calling `AEGetDescDataSize` (page 434).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Your application can call `AEGetDescDataSize` (page 434) to get the size, in bytes, of the data in a descriptor, allocate a buffer or variable of that size, then call `AEGetDescData` to get the data.

This function works only with value descriptors created by `AECreateDesc` (page 417). You cannot get the data of an `AERecord` (page 557) or `AEDescList` (page 553), for example.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`AEDataModel.h`

AEGetDescDataRange

Retrieves a specified series of bytes from the specified descriptor.

```
OSStatus AEGetDescDataRange (
   const AEDesc *dataDesc,
   void *buffer,
   Size offset,
   Size length
);
```

Parameters

dataDesc

A pointer to the descriptor to get the data from. See `AEDesc` (page 546).

buffer

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes should be at least as large as the value you pass in the `length` parameter. On return, contains the specified data from the descriptor.

offset

The zero-based offset to the data to be retrieved from the descriptor.

length

The number of bytes of contiguous data to retrieve.

Return Value

A result code. If the requested `offset` and `length` are such that they do not fit entirely within the descriptor's data, `AEGetDescDataRange` returns `errAEBufferTooSmall`. See also "Apple Event Manager Result Codes" (page 636).

Discussion

This function is valid only for value type descriptors (such as `typeUTF8Text`). You can use this function when you know the precise location of a subset of data within the descriptor. For example, if the descriptor contains a block of your private data, you might retrieve just a particular chunk you need at a known offset, representing an image, a string, or some other data type. Or if a descriptor contains an RGB color, you can access just the blue field.

When used in conjunction with `AECreateDescFromExternalPtr` (page 418), `AEGetDescDataRange` can provide greatly improved performance, especially when working with large blocks of data.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`AEDataModel.h`

AEGetDescDataSize

Gets the size, in bytes, of the data in the specified descriptor.

```
Size AEGetDescDataSize (
   const AEDesc *theAEDesc
);
```

Parameters

theAEDesc

A pointer to the descriptor to obtain the data size for. See `AEDesc` (page 546).

Return Value

Returns the size, in bytes, of the data in the specified descriptor.

Discussion
This function works only with value descriptors created by `AECreateDesc` (page 417). You cannot get the data size of an `AERecord` (page 557) or `AEDescList` (page 553), for example.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetEventHandler

Gets an event handler from an Apple event dispatch table.

```
OSErr AEGetEventHandler (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   AEEventHandlerUPP *handler,
   SRefCon *handlerRefcon,
   Boolean isSysHandler
);
```

Parameters

theAEEventClass

The event class for the desired handler. See `AEEventClass` (page 555).

theAEEventID

The event ID for the desired handler. See `AEEventID` (page 556).

handler

A universal procedure pointer. On return, a pointer to the specified handler, if a dispatch table entry exists that exactly matches the values supplied in the parameters `theAEEventClass` and `theAEEventID`.

If you use the `typeWildCard` constant for either or both of these parameters, `AEGetEventHandler` will return an error unless an entry exists that specifies `typeWildCard` in exactly the same way. For example, if you specify `typeWildCard` in both the `theAEEventClass` parameter and the `theAEEventID` parameter, the Apple Event Manager will not return the first handler for any event class and event ID in the dispatch table; instead, it will only return a handler if an entry exists that specifies type `typeWildCard` for both the event class and the event ID.

For an explanation of wildcard values, see the Discussion section for `AEInstallEventHandler` (page 449).

See `AEEventHandlerUPP` (page 555).

handlerRefcon

A pointer to a reference constant. On return, the reference constant from the dispatch table entry for the specified handler. The reference constant may have a value of 0.

isSysHandler

Specifies the Apple event dispatch table to get the handler from. Pass `TRUE` to get the handler from the system dispatch table or `FALSE` to get the handler from your application's dispatch table. See Version Notes for related information.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

In Mac OS 7.1 through 9.x and Mac OS X version v10.2 and later, `AEGetEventHandler` returns `errAEHandlerNotInstalled` when there's not an exact match, even if a wildcard handler is installed that could handle the event. Mac OS X version v10.0.x and v10.1.x will return the wildcard handler.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEGetInteractionAllowed

Gets your application's current user interaction preferences for responding to an Apple event as a server application.

```
OSErr AEGetInteractionAllowed (
   AEInteractAllowed *level
);
```

Parameters

level

A pointer to an interaction level variable. On return, the variable specifies the current user interaction level, matching one of the values described in "User Interaction Level Constants" (page 605).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The current user interaction preference for responding to an Apple event is set either by default (to `kAEInteractWithLocal`) or by a previous call to `AESetInteractionAllowed` (page 479).

For additional information on interaction level, see `AESend` (page 476) and `"AESendMode"` (page 566).

See also `AEInteractWithUser` (page 453).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEGetKeyDesc

Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event record

```
OSErr AEGetKeyDesc (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    DescType desiredType,
    AEDesc *result
);
```

Parameters

theAERecord

A pointer to the Apple event record to get the parameter descriptor from.

theAEKeyword

A keyword that specifies the desired Apple event parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 595). See `AEKeyword` (page 556).

desiredType

The descriptor type for the desired Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If the requested Apple event parameter is not of the desired type, the Apple Event Manager attempts to coerce it to the desired type. However, if you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the returned descriptor is the same as the descriptor type of the Apple event parameter.

See `DescType` (page 560).

result

A pointer to a descriptor. On successful return, a copy of the descriptor for the specified Apple event parameter, coerced, if necessary, to the descriptor type specified by the `desiredType` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This function is declared as a macro that invokes `AEGetParamDesc` (page 443), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See `AEGetParamDesc` (page 443).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetKeyPtr

Gets a copy of the data for a specified Apple event parameter from an Apple event record.

```
OSErr AEGetKeyPtr (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    DescType desiredType,
    DescType *actualType,
    void *dataPtr,
    Size maximumSize,
    Size *actualSize
);
```

Parameters

theAERecord

A pointer to the Apple event record to get the parameter data from.

theAEKeyword

The keyword that specifies the desired Apple event record parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 595).

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If the descriptor specified by the `theAEKeyword` parameter is not of the desired type, `AEGetKeyPtr` attempts to coerce the data to this type. However, if the desired type is `typeWildCard`, no coercion is performed.

On return, you can determine the actual descriptor type by examining the `typeCode` parameter.

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the data pointed to by `dataPtr`. The returned type is either the same as the type specified by the `desiredType` parameter or, if the desired type was type wildcard, the true type of the descriptor. Specify `NULL` if you do not care about this return value. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the parameter data. Specify `NULL` if you do not care about this return value.

maximumSize

The maximum length, in bytes, of the expected Apple event record parameter data. The `AEGetKeyPtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a variable of type `Size`. On return, the length, in bytes, of the data for the specified Apple event record parameter. If this value is larger than the value you passed in the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the parameter, though `AEGetKeyPtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetKeyPtr` again. Specify `NULL` if you do not care about this return value.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

This function is declared as a macro that invokes `AEGetParamPtr` (page 444), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See AEGetParamPtr (page 444).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEGetNthDesc

Copies a descriptor from a specified position in a descriptor list into a specified descriptor; typically used when your application needs to pass the extracted data to another function as a descriptor.

```
OSErr AEGetNthDesc (
   const AEDescList *theAEDescList,
   long index,
   DescType desiredType,
   AEKeyword *theAEKeyword,
   AEDesc *result
);
```

Parameters

theAEDescList

A pointer to the descriptor list to get the descriptor from. See AEDescList (page 553).

index

A one-based positive integer indicating the position of the descriptor to get. AEGetNthDesc returns an error if you pass zero, a negative number, or a value that is out of range.

desiredType

The desired descriptor type for the descriptor to copy. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If the descriptor specified by the index parameter is not of the desired type, AEGetNthDesc attempts to coerce it to this type. However, if you pass a value of typeWildCard, no coercion is performed, and the descriptor type of the copied descriptor is the same as the descriptor type of the original descriptor.

See DescType (page 560).

theAEKeyword

A pointer to a keyword. On successful return, the keyword for the specified descriptor, if you are getting data from a list of keyword-specified descriptors; otherwise, AEGetNthDesc returns the value typeWildCard. Some keyword constants are described in "Keyword Attribute Constants" (page 593) and "Keyword Parameter Constants" (page 595). See AEKeyword (page 556).

result

A pointer to a descriptor. On successful return, a copy of the descriptor specified by the index parameter, coerced, if necessary, to the descriptor type specified by the desiredType parameter. On error, a null descriptor. If the function returns successfully, your application should call the AEDisposeDesc (page 424) function to dispose of the resulting descriptor after it has finished using it. See AEDesc (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
If the Nth descriptor in the list is itself an Apple event record and the desired type is not wildcard, record, or list, `AEGetNthDesc` will fail with an `errAECoercionFailed` error. This behavior prevents coercion problems.

You may find the `AEGetNthPtr` (page 440) function convenient for retrieving data for direct use in your application, as it includes automatic coercion.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetNthPtr

Gets a copy of the data from a descriptor at a specified position in a descriptor list; typically used when your application needs to work with the extracted data directly.

```
OSErr AEGetNthPtr (
   const AEDescList *theAEDescList,
   long index,
   DescType desiredType,
   AEKeyword *theAEKeyword,
   DescType *typeCode,
   void *dataPtr,
   Size maximumSize,
   Size *actualSize
);
```

Parameters

theAEDescList

A pointer to the descriptor list that contains the descriptor. See `AEDescList` (page 553).

index

A one-based positive integer indicating the position in the descriptor list of the descriptor to get the data from. `AEGetNthPtr` returns an error if you pass zero, a negative number, or a value that is out of range.

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If the descriptor specified by the `index` parameter is not of the desired type, `AEGetNthPtr` attempts to coerce the data to this type. If you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the copied data is the same as the descriptor type of the original descriptor.

See `DescType` (page 560).

theAEKeyword

A pointer to a keyword. On return, the keyword for the specified descriptor, if you are getting data from a list of keyword-specified descriptors; otherwise, `AEGetNthPtr` returns the value `typeWildCard`. Some keyword constants are described in "Keyword Attribute Constants" (page 593) and "Keyword Parameter Constants" (page 595). See `AEKeyword` (page 556).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the data pointed to by `dataPtr`. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the data from the descriptor at the position in the descriptor list specified by the `index` parameter.

maximumSize

The maximum length, in bytes, of the expected data. The `AEGetNthPtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a size variable. On return, the length, in bytes, of the data for the specified descriptor. If this value is larger than the value of the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the descriptor, though `AEGetNthPtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetNthPtr` again.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The `AEGetNthPtr` function uses a buffer to return the data for a specified descriptor from a specified descriptor list. The function attempts to coerce the descriptor to the descriptor type specified by the `desiredType` parameter.

Before calling `AEGetNthPtr`, you can call the `AESizeOfNthItem` (page 484) function to determine a size for the `dataPtr` buffer. However, unless you specify `typeWildCard` for the `desiredType` parameter, `AESizeOfNthItem` may coerce the data, which may cause the size of the data to change. If you are using `AEGetNthPtr` to iterate through a list of descriptors of the same type with a fixed size, such as a list of descriptors of type `typeFSS`, you can get the size once, allocate a buffer, and reuse it for each call.

The order of items in an Apple event record may change after an insertion or deletion. In addition, duplicating an Apple event record is not guaranteed to preserve the item order.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell
QTMetaData

Declared In
`AEDataModel.h`

AEGetObjectAccessor

Gets an object accessor function from an object accessor dispatch table.

```
OSErr AEGetObjectAccessor (
   DescType desiredClass,
   DescType containerType,
   OSLAccessorUPP *accessor,
   SRefCon *accessorRefcon,
   Boolean isSysHandler
);
```

Parameters

desiredClass

The object class of the Apple event objects located by the object accessor function to get. Pass the value `typeWildCard` to get an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the object class. Pass the value `cProperty` to get an object accessor function whose entry in an object accessor dispatch table specifies `cProperty` (a constant used to specify a property of any object class). Some other possible values are defined in "Object Class ID Constants" (page 599). See `DescType` (page 560).

containerType

The descriptor type of the token that identifies the container for the objects located by the requested accessor function. (Token is defined in `AEDisposeToken` (page 425).) Pass the value `typeWildCard` to get an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the descriptor type of the token used to specify the container type. See `DescType` (page 560).

accessor

A universal procedure pointer. On return, a pointer to the requested object accessor function, if an object accessor dispatch table entry exists that exactly matches the values supplied in the parameters `desiredClass` and `containerType`. See `OSLAccessorUPP` (page 560).

accessorRefcon

A pointer to a reference constant. On return, points to the reference constant from the object accessor dispatch table entry for the specified object accessor function. The reference constant may have a value of 0.

isSysHandler

Specifies the object accessor dispatch table to get the object accessor function from. Pass `TRUE` to get the object accessor function from the system object accessor dispatch table or `FALSE` to get the object accessor function from your application's object accessor dispatch table. Use of the system object accessor dispatch table is not recommended.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

Calling `AEGetObjectAccessor` does not remove the object accessor function from an object accessor dispatch table.

Version Notes

In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEObjects.h`

AEGetParamDesc

Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event or an Apple event record.

```
OSErr AEGetParamDesc (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType desiredType,
   AEDesc *result
);
```

Parameters

theAppleEvent

A pointer to the Apple event to get the parameter descriptor from.

theAEKeyword

A keyword that specifies the desired Apple event parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 595).

desiredType

The descriptor type for the desired Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If the requested Apple event parameter is not of the desired type, the Apple Event Manager attempts to coerce it to the desired type. However, if you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the returned descriptor is the same as the descriptor type of the Apple event parameter.

result

A pointer to a descriptor. On successful return, a copy of the descriptor for the specified Apple event parameter, coerced, if necessary, to the descriptor type specified by the `desiredType` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of the resulting descriptor after it has finished using it.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

You typically call `AEGetParamDesc` to get a descriptor for an Apple event parameter to pass on to another Apple Event Manager routine. To get Apple event parameter data for your application to use directly, call `AEGetParamPtr` (page 444).

If the actual parameter you are getting with `AEGetParamDesc` is a record, you can only request it as a `typeAERecord`, `typeAEList`, or `typeWildcard`. For any other type, `AEGetParamDesc` will return `errAECoercionFail`.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

Declared In
`AEDataModel.h`

AEGetParamPtr

Gets a copy of the data for a specified Apple event parameter from an Apple event or an Apple event record.

```
OSErr AEGetParamPtr (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType desiredType,
   DescType *actualType,
   void *dataPtr,
   Size maximumSize,
   Size *actualSize
);
```

Parameters

theAppleEvent

A pointer to the Apple event to get the parameter data from.

theAEKeyword

The keyword that specifies the desired Apple event parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 595).

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If the descriptor specified by the `theAEKeyword` parameter is not of the desired type, `AEGetParamPtr` attempts to coerce the data to this type. However, if the desired type is `typeWildCard`, no coercion is performed.

On return, you can determine the actual descriptor type by examining the `typeCode` parameter.

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the data pointed to by `dataPtr`. The returned type is either the same as the type specified by the `desiredType` parameter or, if the desired type was type wildcard, the true type of the descriptor. Specify `NULL` if you do not care about this return value. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the parameter data. Specify `NULL` if you do not care about this return value.

maximumSize

The maximum length, in bytes, of the expected Apple event parameter data. The `AEGetParamPtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a variable of type `Size`. On return, the length, in bytes, of the data for the specified Apple event parameter. If this value is larger than the value you passed in the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the parameter, though `AEGetParamPtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetParamPtr` again. Specify `NULL` if you do not care about this return value.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
You should use this function only to extract data from value descriptors such as `typeUTF8Text`.

Because this function allows you to specify a desired type, it can result in coercion. When used correctly, this has the positive effect of returning the data in the desired format. However, it can have side effects you may not be expecting, such as the overhead of calls to coercion handlers. See also the Version Notes section below for possible problems with coercion.

To get Apple event parameter data for your application to use directly, call `AEGetParamPtr`. To get a descriptor for an Apple event parameter to pass on to another Apple Event Manager routine, call `AEGetParamDesc` (page 443).

Before calling `AEGetParamPtr`, you can call the `AESizeOfParam` (page 485) function to determine a size for the `dataPtr` buffer. However, unless you specify `typeWildCard` for the `desiredType` parameter, `AEGetParamPtr` may coerce the data, which may cause the size of the data to change.

In some cases, you may get improved efficiency extracting information from an Apple event with the `AEGetDescDataRange` (page 433) function.

Version Notes
Thread safe starting in Mac OS X v10.2.

If the actual parameter you are getting with `AEGetParamPtr` is a record, `AEGetParamPtr` will erroneously allow you to get the parameter as any type at all, when it really should allow only `typeAERecord`, `typeAEList`, or `typeWildcard`. For other types, it will place raw record data into the designated buffer. With AppleScript 1.1.2, it would then return `errAECoercionFail`, as expected. With AppleScript 1.3 and later, however, it returns `noErr`.

You can work around this problem by checking the returned parameter from any call to `AEGetParamPtr`. If the source type is `typeAERecord` and the type you asked for was anything other than `typeAERecord`, `typeAEList`, or `typeWildcard`, you should assume the coercion failed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetRegisteredMachPort

Returns the Mach port (in the form of a `mach_port_t`) that was registered with the bootstrap server for this process.

```
mach_port_t AEGetRegisteredMachPort (
   void
);
```

Return Value
Returns a Mach message port header.

Discussion
Apple events on Mac OS X are implemented in terms of Mach messages. If your application links with the Carbon umbrella framework, it includes the HIToolbox framework, which initializes a Mach port and registers it with the run loop for the application. That port is considered public, and is used for sending and receiving Apple events.

Linking with the HIToolbox also requires that the application have a connection to the window server. To facilitate writing server processes that can send and receive Apple events, the Apple Event Manager provides the following functions (on Mac OS X only): `AEGetRegisteredMachPort`, `AEDecodeMessage` (page 421), `AESendMessage` (page 478), and `AEProcessMessage` (page 458). Daemons and other processes with no user interface can take advantage of these functions, while typical high-level applications will have no need for them.

If your code doesn't link with the HIToolbox or doesn't have a run loop, it can call `AEGetRegisteredMachPort` to register a port directly, then listen on that port for Apple events. It can use the other low-level functions to process incoming Apple events on the port and to send Apple events through it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEMach.h`

AEGetSpecialHandler

Gets a specified handler from a special handler dispatch table.

```
OSErr AEGetSpecialHandler (
   AEKeyword functionClass,
   AEEventHandlerUPP *handler,
   Boolean isSysHandler
);
```

Parameters

`functionClass`

The keyword for the special handler to get. You can specify any of the constants described in "Special Handler Callback Constants" (page 603). See `AEKeyword` (page 556).

`handler`

A universal procedure pointer. On return, a pointer to the specified special handler, if one exists that matches the value supplied in the `functionClass` parameter. See `AEEventHandlerUPP` (page 555).

`isSysHandler`

Specifies the special handler dispatch table to get the handler from. Pass `TRUE` to get the handler from the system special handler dispatch table or `FALSE` to get the handler from your application's special handler dispatch table. Use of the system special handler dispatch table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See also `AEInstallSpecialHandler` (page 452) and `AERemoveSpecialHandler` (page 471).

Version Notes
Thread safe starting in Mac OS X v10.2.

In Mac OS X, you should generally install all handlers in the application dispatch table. For Carbon applications running in Mac OS 8 or Mac OS 9, a special handler in the system dispatch table could reside in the system heap, where it would be available to other applications. However, this won't work in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEGetTheCurrentEvent

Gets the Apple event that is currently being handled.

```
OSErr AEGetTheCurrentEvent (
   AppleEvent *theAppleEvent
);
```

Parameters

`theAppleEvent`

A pointer to an Apple event. On return, the Apple event that is currently being handled. If no Apple event is currently being handled, `AEGetTheCurrentEvent` supplies a descriptor of descriptor type `typeNull`, which does not contain any data. See `AppleEvent` (page 559).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
In many applications, the handling of an Apple event involves one or more long chains of calls to internal functions. The `AEGetTheCurrentEvent` function makes it unnecessary for these calls to include the current Apple event as a parameter; the functions can simply call `AEGetTheCurrentEvent` to get the current Apple event when it is needed.

You can also use the `AEGetTheCurrentEvent` function to make sure that no Apple event is currently being handled. For example, suppose your application always uses an application-defined function to delete a file. That function can first call `AEGetTheCurrentEvent` and delete the file only if `AEGetTheCurrentEvent` returns a null descriptor (that is, only if no Apple event is currently being handled).

Special Considerations

This function is not thread-safe and should only be called on the main thread.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEInitializeDesc

Initializes a new descriptor.

```
void AEInitializeDesc (
   AEDesc *desc
);
```

Parameters

desc

A pointer to a new descriptor. See AEDesc (page 546).

Discussion

The function sets the type of the descriptor to typeNull and sets the data handle to NULL. If you need to initialize a descriptor that already has some data in it, use AEDisposeDesc (page 424) to deallocate the memory and initialize the descriptor.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

AEDataModel.h

AEInstallCoercionHandler

Installs a coercion handler in either the application or system coercion handler dispatch table.

```
OSErr AEInstallCoercionHandler (
   DescType fromType,
   DescType toType,
   AECoercionHandlerUPP handler,
   SRefCon handlerRefcon,
   Boolean fromTypeIsDesc,
   Boolean isSysHandler
);
```

Parameters

fromType

The descriptor type of the data coerced by the handler. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See DescType (page 560).

toType

The descriptor type of the resulting data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581).

If there was already an entry in the specified coercion handler table for the same source descriptor type and result descriptor type, the existing entry is replaced. See DescType (page 560).

handler

A universal procedure pointer to the coercion handler function to install. See AECoercionHandlerUPP (page 552).

handlerRefcon

A reference constant. The Apple Event Manager passes this value to the handler each time it calls it. If your handler doesn't require a reference constant, pass 0 for this parameter.

fromTypeIsDesc

Specifies the form of the data to coerce. Pass `TRUE` if the coercion handler expects the data as a descriptor or `FALSE` if the coercion handler expects a pointer to the data. The Apple Event Manager can provide a pointer to data more efficiently than it can provide a descriptor, so all coercion functions should accept a pointer to data if possible.

isSysHandler

Specifies the coercion table to add the handler to. Pass `TRUE` to add the handler to the system coercion table or `FALSE` to add the handler to your application's coercion table. Use of the system coercion table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Before using `AEInstallCoercionHandler` to install a handler for a particular descriptor type, you can use the `AEGetCoercionHandler` (page 431) function to determine whether the table already contains a coercion handler for that type.

Version Notes
See the Version Notes section for the `AECoercePtr` (page 414) function for information on when to use descriptor-based versus pointer-based coercion handlers starting in Mac OS X version 10.2.

Thread safe starting in Mac OS X v10.2.

Your application should not install a coercion handler in a system coercion handler dispatch table with the goal that the handler will get called when other applications perform coercions—this won't work in Mac OS X. For more information, see "Writing and Installing Coercion Handlers" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEInstallEventHandler

Adds an entry for an event handler to an Apple event dispatch table.

```
OSErr AEInstallEventHandler (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   AEEventHandlerUPP handler,
   SRefCon handlerRefcon,
   Boolean isSysHandler
);
```

Parameters

theAEEventClass

The event class for the Apple event or events to dispatch to this event handler. The Discussion section describes interactions between this parameter and the `theAEEventID` parameter. See `AEEventClass` (page 555).

theAEEventID

The event ID for the Apple event or events to dispatch to this event handler. The Discussion section describes interactions between this parameter and the `theAEEventClass` parameter. See `AEEventID` (page 556).

handler

A universal procedure pointer to the Apple event handler function to install. See `AEEventHandlerUPP` (page 555).

handlerRefcon

A reference constant. The Apple Event Manager passes this value to the handler each time it calls it. If your handler doesn't require a reference constant, pass 0 for this parameter.

isSysHandler

Specifies the Apple event dispatch table to add the handler to. Pass `TRUE` to add the handler to the system dispatch table or `FALSE` to add the handler to your application's dispatch table. See Version Notes for related information.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

The parameters `theAEEventClass` and `theAEEventID` specify the event class and event ID of the Apple events handled by the handler for this dispatch table entry. If there is already an entry in the specified dispatch table for the same event class and event ID, it is replaced. For these parameters, you must provide one of the following combinations:

- the event class and event ID of a single Apple event to dispatch to the handler (for example, an event class of `kAECoreSuite` and an event ID of `kAEDelete` so that a specific kind of `delete` event is dispatched to the handler)
- the `typeWildCard` constant for `theAEEventClass` and an event ID for `theAEEventID`, which indicates that Apple events from all event classes whose event IDs match `theAEEventID` should be dispatched to the handler (for example, an event class of `typeWildCard` and an event ID of `kAEDelete` so that for all event classes, the `delete` event is dispatched to the handler)
- an event class for `theAEEventClass` and the `typeWildCard` constant for `theAEEventID`, which indicates that all events from the specified event class should be dispatched to the handler (for example, an event class of `kAECoreSuite` and an event ID of `typeWildCard` so that all events for the core suite are dispatched to the handler)
- the `typeWildCard` constant for both the `theAEEventClass` and `theAEEventID` parameters, which indicates that all Apple events should be dispatched to the handler

If you use the `typeWildCard` constant for either the `theAEEventClass` or the `theAEEventID` parameter (or for both parameters), the corresponding handler must return the error `errAEEventNotHandled` if it does not handle a particular event.

If an Apple event dispatch table contains one entry for an event class and a specific event ID, and also contains another entry that is identical except that it specifies a wildcard value for either the event class or the event ID, the Apple Event Manager dispatches the more specific entry. For example, if an Apple event dispatch table includes one entry that specifies the event class as `kAECoreSuite` and the event ID as `kAEDelete`, and another entry that specifies the event class as `kAECoreSuite` and the event ID as `typeWildCard`, the Apple Event Manager dispatches the Apple event handler associated with the entry that specifies the event ID as `kAEDelete`.

In addition to the Apple event handler dispatch tables, applications can add entries to special handler dispatch tables, as described in "Managing Special Handler Dispatch Tables" (page 404).

Version Notes
Thread safe starting in Mac OS X v10.2.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`AppleEvents.h`

AEInstallObjectAccessor

Adds or replaces an entry for an object accessor function to an object accessor dispatch table.

```
OSErr AEInstallObjectAccessor (
   DescType desiredClass,
   DescType containerType,
   OSLAccessorUPP theAccessor,
   SRefCon accessorRefcon,
   Boolean isSysHandler
);
```

Parameters

desiredClass
The object type of the Apple event objects located by this accessor. See `DescType` (page 560).

containerType
The type of the token whose objects are accessed by this accessor. (Token is defined in `AEDisposeToken` (page 425).) The accessor function finds objects in containers specified by tokens of this type. See `DescType` (page 560).

theAccessor
A universal procedure pointer to the object accessor function to install. See `OSLAccessorUPP` (page 560).

accessorRefcon
A reference constant the Apple Event Manager passes to the object accessor function whenever it calls the function. If your object accessor function doesn't require a reference constant, pass 0 for this parameter. To change the value of the reference constant, you must call `AEInstallObjectAccessor` again.

isSysHandler
Specifies the object accessor dispatch table to add the entry to. Pass `TRUE` to add the entry to the system object accessor dispatch table or `FALSE` to add the entry to your application's object accessor dispatch table. Use of the system object accessor dispatch table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The `AEInstallObjectAccessor` function adds or replaces an entry to either the application or system object accessor dispatch table.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

If your Carbon application running in Mac OS 8 or OS 9 installs a system object accessor function in its application heap, rather than in the system heap, you must call `AERemoveObjectAccessor` (page 470) to remove the function before your application terminates.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEInstallSpecialHandler

Installs a callback function in a special handler dispatch table.

```
OSErr AEInstallSpecialHandler (
   AEKeyword functionClass,
   AEEventHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

functionClass

A value that specifies the type of handler to install. You can use any of the constants defined in "Special Handler Callback Constants" (page 603).

If there is already an entry in the specified special handler dispatch table for the value you specify in this parameter, it is replaced.

See `AEKeyword` (page 556).

handler

A universal procedure pointer to the special handler to install. See `AEEventHandlerUPP` (page 555).

isSysHandler

Specifies the special handler dispatch table to add the handler to. Pass `TRUE` to add the handler to the system special handler dispatch table or `FALSE` to add the handler to your application's special handler dispatch table. Use of the system special handler dispatch table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
An Apple event special handler dispatch table contains entries with a function class keyword, the address of the handler function that handles the Apple events indicated by the keyword, and a reference constant. Depending on which handlers you choose to install, a special handler dispatch table can have entries for any of the following:

- a predispatch handler (an Apple event handler that the Apple Event Manager calls immediately before it dispatches an Apple event)
- up to one each of the callback functions described in "Object Callback Functions" (page 524) these functions, such as an object comparison function and an object-counting function, can be installed with `AEInstallSpecialHandler` or with the `AEInstallObjectAccessor` (page 451) function

See also `AEGetSpecialHandler` (page 446) and `AERemoveSpecialHandler` (page 471).

Version Notes
Thread safe starting in Mac OS X v10.2.

For Carbon applications running in Mac OS 8 or Mac OS 9, a handler in the system special handler dispatch table should reside in the system heap, where it may be available to other applications. If you put your system handler code in your application heap, be sure to use `AERemoveSpecialHandler` to remove the handler when your application quits. Otherwise, your handler will still have an entry in the system dispatch table with a pointer a handler that no longer exists. Another application may dispatch an Apple event that attempts to call your handler, leading to a system crash.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEInteractWithUser

Initiates interaction with the user when your application is a server application responding to an Apple event.

```
OSErr AEInteractWithUser (
   SInt32 timeOutInTicks,
   NMRecPtr nmReqPtr,
   AEIdleUPP idleProc
);
```

Parameters

timeOutInTicks
The amount of time (in ticks) that your handler is willing to wait for a response from the user. You can specify a number of ticks or use one of the constants defined in "Timeout Constants" (page 605).

nmReqPtr
A pointer to a Notification Manager record provided by your application. You can specify `NULL` for this parameter to get the default notification handling provided by the Apple Event Manager. See the Notification Manager documentation for a description of the `NMRecPtr` data type.

idleProc
A universal procedure pointer to your application's idle function, which handles events while waiting for the Apple Event Manager to return control. See `AEIdleUPP` (page 556).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). The `AEInteractWithUser` function returns the `errAENoUserInteraction` result code if the user interaction preferences don't allow user interaction. If `AEInteractWithUser` returns the `noErr` result code, then your application is in the foreground and is free to interact with the user.

Discussion
Your application should call the `AEInteractWithUser` function before displaying a dialog box or alert box or otherwise interacting with the user in response to an Apple event. The `AEInteractWithUser` function checks whether the client application set the `kAENeverInteract` flag for the current Apple event, if any, and if so, returns an error. If not, then `AEInteractWithUser` checks the server application's preference set by the `AESetInteractionAllowed` (page 479) function and compares it against the source of the Apple event—that is, whether it came from the same application, another process on the same computer, or a process running on another computer.

If the user interaction preference settings permit the application to come to the foreground, this function brings your application to the front, either directly or by posting a notification request.

Your application should normally pass a notification record in the `nmReqPtr` parameter rather than specifying `NULL` for default notification handling. If you specify `NULL`, the Apple Event Manager looks for an application icon with the ID specified by the application's bundle (`'BNDL'`) resource and the application's file reference (`'FREF'`) resource. The Apple Event Manager first looks for an `'SICN'` resource with the specified ID if it can't find an `'SICN'` resource, it looks for the `'ICN#'` resource and compresses the icon to fit in the menu bar. The Apple Event Manager won't look for any members of an icon family other than the icon specified in the `'ICN#'` resource.

If the application doesn't have `'SICN'` or `'ICN#'` resources, or if it doesn't have a file reference resource, the Apple Event Manager passes no icon to the Notification Manager, and no icon appears in the upper-right corner of the screen. Therefore, if you want to display any icon other than those of type `'SICN'` or `'ICN#'`, you must specify a notification record as the second parameter to the AEInteractWithUser function.

If you want the Notification Manager to use a color icon when it posts a notification request, you should provide a Notification Manager record that specifies a `'cicn'` resource.

For additional information on interaction level, see `AESend` (page 476) and "`AESendMode`" (page 566).

See also `AESetInteractionAllowed` (page 479) and `AEGetInteractionAllowed` (page 436).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEManagerInfo

Provides information about the version of the Apple Event Manager currently available or the number of processes that are currently recording Apple events.

```
OSErr AEManagerInfo (
   AEKeyword keyWord,
   long *result
);
```

Parameters

keyWord

A value that determines the kind of information the function supplies in the `result` parameter.

Pass the value `keyAERecorderCount` to obtain the number of processes that are currently recording Apple events.

Pass the value `keyAEVersion` to obtain version information for the Apple Event Manager, in `NumVersion` format.

Some keyword constants are defined in "Keyword Parameter Constants" (page 595).

See `AEKeyword` (page 556).

result

A pointer to a long value. On return, provides information that depends on what you pass in the `keyword` parameter.

If you pass `keyAERecorderCount`, `result` specifies the number of processes that are currently recording Apple events.

If you pass `keyAEVersion`, `result` supplies version information for the Apple Event Manager, in a format that matches the `'vers'` resource.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
For recordable applications, the information provided by `AEManagerInfo` may be useful when the application is responding to Apple events that it sends to itself.

For information on determining whether the Apple Event Manager is available, see the Apple Event Manager Gestalt Selector, described in *Inside Mac OS X: Gestalt Manager Reference*.

Version Notes
Thread safe starting in Mac OS X v10.2.

The `AEManagerInfo` function is available only in version 1.01 and later of the Apple Event Manager.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEObjectInit

Initializes the Object Support Library.

```
OSErr AEObjectInit (
   void
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
You must call this function before calling any of the Apple Event Manager functions that describe or manipulate Apple event objects.

You should call the `AEObjectInit` function to initialize the Apple Event Manager functions that handle object specifiers and Apple event objects.

Version Notes
To make these functions available to your application with version 1.01 and earlier versions of the Apple Event Manager, you must also link the Apple Event Object Support Library with your application when you build it. For more information, see the Version Notes section for the AppleScript Gestalt Selector described in *Inside Mac OS X: Gestalt Manager Reference* and the function `AERemoveSpecialHandler` (page 471).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEPrintDescToHandle

Provides a pretty printer facility for displaying the contents of Apple event descriptors.

```
OSStatus AEPrintDescToHandle (
   const AEDesc *desc,
   Handle *result
);
```

Parameters

desc

A pointer to a descriptor containing the information to be printed. See `AEDesc` (page 546).

result

A pointer to a location for a new `Handle` data type. On return, contains a new handle allocated by the Memory Manager.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The data handle returned in the *result* parameter contains a text string formatted using the "AEBuild" syntax. This string is useful for looking at the contents of Apple events sent by other applications and for debugging your own descriptors.

`AEPrintDescToHandle` prints the contents of `AEDesc`, `AERecord`, and `AEDescList` descriptors in a format that is suitable for input to `AEBuildDesc` (page 410). `AEPrintDescToHandle` also attempts display coerced Apple event records as the coerced record type instead of as the original type. Any data structures that cannot be identified are displayed as hexadecimal data.

`AEPrintDescToHandle` prints the contents of Apple events in a slightly different format. For these events, the event class and event ID appear at the beginning of the output string, followed by the contents of the event enclosed in curly braces. In addition, each attribute is printed with its four-character identifier and preceded by an ampersand character. You cannot use the output string to recreate the Apple event from `AEBuildAppleEvent` (page 408).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEProcessAppleEvent

Calls the handler, if one exists, for a specified Apple event.

```
OSErr AEProcessAppleEvent (
   const EventRecord *theEventRecord
);
```

Parameters

`theEventRecord`

A pointer to the event record for the Apple event to process. See the Event Manager documentation for a description of the `EventRecord` data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). This is the error result from the Apple event handler (or `errAEHandlerNotFound`). In most cases your application should ignore this error because it will be seen by the Apple event sender as the `keyErrorNumber` parameter in the reply.

Discussion
After receiving a high-level event (and optionally determining whether it is a type of high-level event other than an Apple event that your application might support), your application typically calls the `AEProcessAppleEvent` function to determine the type of Apple event received and call the corresponding handler. Your application should always handle high-level events immediately, or the Apple Event Manager may return the event to the sending application with the `errAEEventNotHandled` result code.

The `AEProcessAppleEvent` function looks first in the application's special handler dispatch table for an entry that was installed by the `AEInstallSpecialHandler` (page 452) function with the constant `keyPreDispatch`. If the application's special handler dispatch table does not include such a handler or if the handler returns `errAEEventNotHandled`, `AEProcessAppleEvent` looks in the application's Apple event dispatch table for an entry that matches the event class and event ID of the specified Apple event. You install handlers in the application's dispatch table with the `AEInstallEventHandler` (page 449) function.

If the application's Apple event dispatch table does not include such a handler or if the handler returns `errAEEventNotHandled`, the `AEProcessAppleEvent` function looks in the system special handler dispatch table for an entry that was installed with the constant `keyPreDispatch`. If the system special handler dispatch table does not include such a handler or if the handler returns `errAEEventNotHandled`, `AEProcessAppleEvent` looks in the system Apple event dispatch table for an entry that matches the event class and event ID of the specified Apple event.

If the system Apple event dispatch table does not include such a handler, the Apple Event Manager returns the result code `errAEEventNotHandled` to the server (or target) application and, if the client application is waiting for a reply, to the client application.

If AEProcessAppleEvent finds an entry in one of the dispatch tables that matches the event class and event ID of the specified Apple event, it calls the corresponding handler.

If an Apple event dispatch table contains one entry for an event class and a specific event ID, and also contains another entry that specifies a wildcard value for either the event class or the event ID, the Apple Event Manager uses the more specific entry. For example, if one entry specifies an event class of kAECoreSuite and an event ID of kAEDelete and another entry specifies an event class of kAECoreSuite and an event ID of typeWildCard, the Apple Event Manager will dispatch an Apple event with an event ID of kAEDelete to the handler from the entry that specifies the event ID as kAEDelete.

Version Notes
Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Simple DrawSprocket

Declared In
AEInteraction.h

AEProcessMessage

Decodes and dispatches a low level Mach message event to an event handler, including packaging and returning the reply to the sender.

```
OSStatus AEProcessMessage (
   mach_msg_header_t *header
);
```

Parameters

header

A pointer to the received Mach message that should be processed. The contents of the message header are invalid after calling this method.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The Apple Event Manager provides the following functions (on Mac OS X only) for working with Apple events at a lower level: AEGetRegisteredMachPort (page 445), AEDecodeMessage (page 421), AESendMessage (page 478), and AEProcessMessage. See the descriptions for those functions for more information on when you might use them.

If your daemon or other code has initialized a Mach port and is listening on it for Apple events and other messages, it can call AEProcessMessage to handle any incoming events it identifies as Apple events, while handling other types of events itself. AEProcessMessage will dispatch the event to an event handler (by calling AEDecodeMessage for you) and package and return the reply to the sender, simplifying handling for your code.

The Apple Event Manager reserves Mach message IDs in the range 0 to 999 for its own use. AEProcessMessage returns a paramErr result code if the Mach message did not contain an Apple event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEMach.h

AEPutArray

Inserts the data for an Apple event array into a descriptor list, replacing any previous descriptors in the list.

```
OSErr AEPutArray (
   AEDescList *theAEDescList,
   AEArrayType arrayType,
   const AEArrayData *arrayPtr,
   DescType itemType,
   Size itemSize,
   long itemCount
);
```

Parameters

theAEDescList

A pointer to the descriptor list to put the Apple event array into. If there are any descriptors already in the descriptor list, they are replaced. If the array type is kAEKeyDescArray, theAEDescList must point to an Apple event record; otherwise, it can point to either a descriptor list or an Apple event record.

If you pass a pointer to a factored descriptor list, created by calling the AECreateList (page 419) function, each array item in the array pointed to by the arrayPtr parameter must include the data that is common to all the descriptors in the list. The Apple Event Manager automatically isolates the common data you specified in the call to AECreateList. A factored descriptor list is described in the Discussion section.

See AEDescList (page 553).

arrayType

The Apple event array type to create. Pass a value specified by one of the constants described in "Data Array Constants" (page 580). See AEArrayType (page 552).

arrayPtr

A pointer to a buffer, local variable, or other storage location, created and disposed of by your application, that contains the array to put into the descriptor list. See AEArrayData (page 545).

itemType

For arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray, the descriptor type of the array items to create. Use one of the constants described in "Descriptor Type Constants" (page 581), such as typeLongInteger. You don't need to specify an item type for arrays of type kAEDescArray or kAEKeyDescArray because the data is already stored in descriptors which contain a descriptor type. See DescType (page 560).

itemSize

For arrays of type kAEDataArray or kAEPackedArray, the size (in bytes) of the array items to create. You don't need to specify an item size for arrays of type kAEDescArray, kAEKeyDescArray, or kAEHandleArray because their descriptors (though not the data they point to) have a known size.

itemCount

The number of elements in the array.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
A factored descriptor list is one in which the Apple Event Manager automatically isolates the data that is common to all the elements of the list so that the common data only appears in the list once. To create a factored descriptor list, you call the `AECreateList` (page 419) function and specify the data that is common to all elements in the descriptor array.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEPutAttributeDesc

Adds a descriptor and a keyword to an Apple event as an attribute.

```
OSErr AEPutAttributeDesc (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   const AEDesc *theAEDesc
);
```

Parameters

theAppleEvent
A pointer to the Apple event to add an attribute to. See the `AppleEvent` (page 559) data type.

theAEKeyword
The keyword for the attribute to add. If the Apple event already includes an attribute with this keyword, the attribute is replaced.

Some keyword constants are described in "Keyword Attribute Constants" (page 593).

See `AEKeyword` (page 556).

theAEDesc
A pointer to the descriptor to assign to the attribute. The descriptor type of the specified descriptor should match the defined descriptor type for that attribute. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The `AEPutAttributeDesc` function takes a descriptor and a keyword and adds them to an Apple event as an attribute. If the descriptor type required for the attribute is different from the descriptor type of the descriptor, the Apple Event Manager attempts to coerce the descriptor into the required type, with one exception: the Apple Event Manager does not attempt to coerce the data for an address attribute, thereby allowing applications to use their own address types.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutAttributePtr

Adds a pointer to data, a descriptor type, and a keyword to an Apple event as an attribute.

```
OSErr AEPutAttributePtr (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType typeCode,
   const void *dataPtr,
   Size dataSize
);
```

Parameters

theAppleEvent
A pointer to the Apple event to add an attribute to. See the `AppleEvent` (page 559) data type.

theAEKeyword
The keyword for the attribute to add. If the Apple event already includes an attribute with this keyword, the attribute is replaced.

Some keyword constants are described in "Keyword Attribute Constants" (page 593).

See `AEKeyword` (page 556).

typeCode
The descriptor type for the attribute to add. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr
A pointer to the data for the attribute to add.

dataSize
The length, in bytes, of the data for the attribute to add.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutDesc

Adds a descriptor to any descriptor list, possibly replacing an existing descriptor in the list.

```
OSErr AEPutDesc (
   AEDescList *theAEDescList,
   long index,
   const AEDesc *theAEDesc
);
```

Parameters

theAEDescList

A pointer to the descriptor list to add a descriptor to. See `AEDescList` (page 553).

index

A one-based positive integer indicating the position to insert the descriptor at. If there is already a descriptor in the specified position, it is replaced.

You can pass a value of zero or count + 1 to add the descriptor at the end of the list. `AEPutDesc` returns an error (`AEIllegalIndex`) if you pass a negative number or a value that is out of range.

theAEDesc

A pointer to the descriptor to add to the list. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEPutKeyDesc

Inserts a descriptor and a keyword into an Apple event record as an Apple event parameter.

```
OSErr AEPutKeyDesc (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    const AEDesc *theAEDesc
);
```

Parameters

theAERecord

A pointer to the Apple event record to add a parameter to.

theAEKeyword

The keyword specifying the parameter to add. If the Apple event record already has a parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 595).

See `AEKeyword` (page 556).

theAEDesc

A pointer to the descriptor for the parameter to add. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This function is declared as a macro that invokes `AEPutParamDesc` (page 464), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See `AEPutParamDesc` (page 464).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEPutKeyPtr

Inserts data, a descriptor type, and a keyword into an Apple event record as an Apple event parameter.

```
OSErr AEPutKeyPtr (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    DescType typeCode,
    const void *dataPtr,
    Size dataSize
);
```

Parameters

theAERecord
A pointer to the Apple event record to add a parameter to.

theAEKeyword
The keyword for the parameter to add. If the Apple event record already includes a parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 595).

See `AEKeyword` (page 556).

typeCode
The descriptor type for the parameter to add. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr
A pointer to the data for the parameter to add.

dataSize
The length, in bytes, of the data for the parameter to add.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This function is declared as a macro that invokes `AEPutParamPtr` (page 464), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See `AEPutParamPtr` (page 464).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutParamDesc

Inserts a descriptor and a keyword into an Apple event or Apple event record as an Apple event parameter.

```
OSErr AEPutParamDesc (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   const AEDesc *theAEDesc
);
```

Parameters

theAppleEvent
A pointer to the Apple event to add a parameter to. See the AppleEvent (page 559) data type.

theAEKeyword
The keyword specifying the parameter to add. If the Apple event already has a parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 595).

See AEKeyword (page 556).

theAEDesc
A pointer to the descriptor for the parameter to add. See AEDesc (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutParamPtr

Inserts data, a descriptor type, and a keyword into an Apple event or Apple event record as an Apple event parameter.

```
OSErr AEPutParamPtr (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType typeCode,
   const void *dataPtr,
   Size dataSize
);
```

Parameters

theAppleEvent
A pointer to the Apple event to add a parameter to. See the AppleEvent (page 559) data type.

theAEKeyword

The keyword for the parameter to add. If the Apple event already includes an parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 595).

See `AEKeyword` (page 556).

typeCode

The descriptor type for the parameter to add. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr

A pointer to the data for the parameter to add.

dataSize

The length, in bytes, of the data for the parameter to add.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEPutPtr

Inserts data specified in a buffer into a descriptor list as a descriptor, possibly replacing an existing descriptor in the list.

```
OSErr AEPutPtr (
   AEDescList *theAEDescList,
   long index,
   DescType typeCode,
   const void *dataPtr,
   Size dataSize
);
```

Parameters

theAEDescList

A pointer to the descriptor list to add a descriptor to. See `AEDescList` (page 553).

index

A one-based positive integer indicating the position to insert the descriptor at. If there is already a descriptor in the specified position, it is replaced.

You can pass a value of zero or count + 1 to add the descriptor at the end of the list. `AEPutPtr` returns an error (`AEIllegalIndex`) if you pass a negative number or a value that is out of range.

typeCode

The descriptor type for the descriptor to be put into the list. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr
A pointer to the data for the descriptor to add.

dataSize
The length, in bytes, of the data for the descriptor to add.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`AEDataModel.h`

AERemoteProcessResolverGetProcesses

Returns an array of objects containing information about processes running on a remote machine.

```
CFArrayRef AERemoteProcessResolverGetProcesses (
   AERemoteProcessResolverRef ref,
   CFStreamError *outError
);
```

Parameters

ref
The `AERemoteProcessResolverRef` (page 557) to query. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 420).

outError
If the function result is `NULL`, `outError` contains information about the failure. See the Core Foundation Reference Documentation for a description of the `CFStreamError` data type.

Return Value
In the case of an error, returns `NULL`, in which case the `outError` parameter provides error information. If successful, returns a `CFArrayRef` of `CFDictionaryRef` objects containing information about the discovered remote processes. Each dictionary contains the URL of a remote application and its human readable name; it may also contain a `CFNumberRef` specifying a user ID for the application, if it has one; and it may also contain a `CFNumberRef` specifying the process ID for the process. The array is owned by the resolver, so you must retain it before disposing of the resolver object itself. For information on the keys for getting information from the dictionary, see "Remote Process Dictionary Keys" (page 602).

Discussion
You first call `AECreateRemoteProcessResolver` (page 420) to obtain a reference to a resolver object you can use to obtain a list of processes running on a specified remote machine. See the description for that function for additional information. You then pass that reference to `AERemoteProcessResolverGetProcesses` to get an array of objects containing information about the discovered remote processes.

If the resolver was not previously scheduled for execution (by a call to the `AERemoteProcessResolverScheduleWithRunLoop` (page 467) function), `AERemoteProcessResolverGetProcesses` will block until the resulting array is available or an error occurs. If the resolver was previously scheduled but had not yet completed fetching the array, this call will block until the resolver does complete.

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AERemoteProcessResolverScheduleWithRunLoop

Schedules a resolver for execution on a given run loop in a given mode.

```
void AERemoteProcessResolverScheduleWithRunLoop (
   AERemoteProcessResolverRef ref,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode,
   AERemoteProcessResolverCallback callback,
   const AERemoteProcessResolverContext *ctx
);
```

Parameters

ref

The `AERemoteProcessResolverRef` (page 557) to query. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 420).

runLoop

The run loop on which to schedule resolution of remote processes. For information on run loops, see Introduction to Run Loops. See the Core Foundation Reference Documentation for a description of the `CFRunLoop` data type.

runLoopMode

Specifies the run loop mode. See Input Modes for information on available modes. See the Core Foundation Reference Documentation for a description of the `CFStringRef` data type.

callback

A callback function to be executed when the resolver completes. See `AERemoteProcessResolverCallback` (page 532) for information on the callback definition.

ctx

Optionally supplies information of use while resolving remote processes. If this parameter is not `NULL`, the info field of this structure is passed to the callback function (otherwise, the info parameter to the `callback` function will explicitly be `NULL`). See `AERemoteProcessResolverContext` (page 547) for a description of this data type.

Discussion
Schedules a resolver for execution on a given run loop in a given mode. The resolver will move through various internal states as long as the specified run loop is run. When the resolver completes, either with success or with an error condition, the callback is executed. There is no explicit unschedule of the resolver; you must dispose of it to remove it from the run loop.

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AERemoveCoercionHandler

Removes a coercion handler from a coercion handler dispatch table.

```
OSErr AERemoveCoercionHandler (
   DescType fromType,
   DescType toType,
   AECoercionHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

fromType

The descriptor type of the data coerced by the handler. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

toType

The descriptor type of the resulting data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

handler

A universal procedure pointer to the coercion handler to remove. Although the parameters `fromType` and `toType` are sufficient to identify the handler, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the event class and event ID to identify the handler. See `AECoercionHandlerUPP` (page 552).

isSysHandler

Specifies the coercion table to remove the handler from. Pass `TRUE` to remove the handler from the system coercion table or `FALSE` to remove the handler from your application's coercion table. Use of the system coercion table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Use of system coercion tables is not recommended. For more information, see "Writing and Installing Coercion Handlers" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERemoveEventHandler

Removes an event handler entry from an Apple event dispatch table.

```
OSErr AERemoveEventHandler (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   AEEventHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

theAEEventClass

The event class for the handler to remove. See `AEEventClass` (page 555).

theAEEventID

The event ID for the handler to remove. See `AEEventID` (page 556).

handler

A universal procedure pointer to the handler to remove. Although the parameters `theAEEventClass` and `theAEEventID` are sufficient to identify the handler, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the event class and event ID to identify the handler.

If you use the `typeWildCard` constant for either or both of the event class and event ID parameters, `AERemoveEventHandler` will return an error unless an entry exists that specifies `typeWildCard` in exactly the same way. For example, if you specify `typeWildCard` in both the `theAEEventClass` parameter and the `theAEEventID` parameter, `AERemoveEventHandler` will not remove the first handler for any event class and event ID in the dispatch table; instead, it will only remove a handler if an entry exists that specifies type `typeWildCard` for both the event class and the event ID.

For an explanation of wildcard values, see the Discussion section for `AEInstallEventHandler` (page 449).

See `AEEventHandlerUPP` (page 555).

isSysHandler

Specifies the Apple event dispatch table to remove the handler from. Pass `TRUE` to remove the handler from the system dispatch table or `FALSE` to remove the handler from your application's dispatch table. See Version Notes for related information.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes

Thread safe starting in Mac OS X v10.2.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AppleEvents.h`

AERemoveObjectAccessor

Removes an object accessor function from an object accessor dispatch table.

```
OSErr AERemoveObjectAccessor (
   DescType desiredClass,
   DescType containerType,
   OSLAccessorUPP theAccessor,
   Boolean isSysHandler
);
```

Parameters

desiredClass

The object class of the Apple event objects located by the object accessor function to remove. Pass the value `typeWildCard` to remove an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the object class. Pass the value `cProperty` to remove an object accessor function whose entry in an object accessor dispatch table specifies `cProperty` (a constant used to specify a property of any object class). Some other possible values are defined in "Object Class ID Constants" (page 599). See `DescType` (page 560).

containerType

The descriptor type of the token that identifies the container for the objects located by the object accessor function to remove. (Token is defined in `AEDisposeToken` (page 425).) Pass the value `typeWildCard` to remove an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the descriptor type of the token used to specify the container type. See `DescType` (page 560).

theAccessor

A universal procedure pointer to the special handler to remove. Although the `functionClass` parameter is sufficient to identify the handler to remove, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the function class to identify the handler. A universal procedure pointer (UPP) to the object accessor function to remove. Although the parameters `desiredClass` and `containerType` are sufficient to identify the function to remove, you can identify the function explicitly by providing a UPP in this parameter. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the desired class and container type. See `OSLAccessorUPP` (page 560).

isSysHandler

Specifies the object accessor dispatch table to remove the object accessor function from. Pass `TRUE` to remove the object accessor function from the system object accessor dispatch table or `FALSE` to remove the object accessor function from your application's object accessor dispatch table. Use of the system object accessor dispatch table is not recommended.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes

In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEObjects.h`

AERemoveSpecialHandler

Removes a handler from a special handler dispatch table.

```
OSErr AERemoveSpecialHandler (
   AEKeyword functionClass,
   AEEventHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

functionClass

The keyword for the special handler to remove. Pass one of the constants described in "Special Handler Callback Constants" (page 603). See `AEKeyword` (page 556).

handler

A universal procedure pointer to the special handler to remove. Although the `functionClass` parameter is sufficient to identify the handler to remove, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the function class to identify the handler. See `AEEventHandlerUPP` (page 555).

isSysHandler

Specifies the special handler dispatch table to remove the handler from. Pass `TRUE` to remove the handler from the system special handler dispatch table or `FALSE` to remove the handler from your application's special handler dispatch table. Use of the system special handler dispatch table is not recommended.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

See also `AEInstallSpecialHandler` (page 452) and `AEGetSpecialHandler` (page 446).

Version Notes

Thread safe starting in Mac OS X v10.2.

Your application should not install a special handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X.

In some previous versions of the Mac OS, applications might have reason to disable, within the application only, all Apple Event Manager functions that support Apple event objects—that is, all the functions available to an application as a result of linking the Object Support Library (OSL) and calling the `AEObjectInit` (page 455) function.

To disable the OSL, you should pass the keyword `keySelectProc` in the `functionClass` parameter, `NULL` in the `handler` parameter, and `FALSE` in the `isSysHandler` parameter. An application that expects its copy of the OSL to move after it is installed—for example, an application that keeps it in a stand-alone code resource—would need to disable the OSL. When an application calls `AEObjectInit` to initialize the OSL, the OSL installs the addresses of its functions as extensions to the pack. If those functions move, the addresses become invalid.

Once you have called the `AERemoveSpecialHandler` function to disable the OSL, subsequent calls by your application to any of the Apple Event Manager functions that support Apple event objects will return errors. To initialize the OSL after disabling it with the `AERemoveSpecialHandler` function, your application must call `AEObjectInit` again.

If you expect to initialize the OSL and disable it several times, you should call `AERemoveObjectAccessor` to remove your application's object accessor functions from your application's object accessor dispatch table before you call `AERemoveSpecialHandler`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEReplaceDescData

Copies the specified data into the specified descriptor, replacing any previous data.

```
OSErr AEReplaceDescData (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   AEDesc *theAEDesc
);
```

Parameters

typeCode
Specifies the descriptor type of the data pointed to by `dataPtr`. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr
A pointer to the data to store in the specified descriptor.

dataSize
The size, in bytes, of the data pointed to by the `dataSize` parameter.

theAEDesc
A pointer to a descriptor. On return, contains the copied data. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEResetTimer

Resets the timeout value for an Apple event to its starting value.

```
OSErr AEResetTimer (
   const AppleEvent *reply
);
```

Parameters

reply

A pointer to the default reply for an Apple event, provided by the Apple Event Manager. See `AppleEvent` (page 559).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

The `AEResetTimer` function resets the timeout value for an Apple event to its starting value. A server application can call this function when it knows it cannot fulfill a client application's request (either by returning a result or by sending back a reply Apple event) before the client application is due to time out.

When your application calls `AEResetTimer`, the Apple Event Manager for the server application uses the default reply to send a Reset Timer event to the client application the Apple Event Manager for the client application's computer intercepts this Apple event and resets the client application's timer for the Apple event. (The Reset Timer event is never dispatched to a handler, so the client application does not need a handler for it.)

Version Notes

Prior to Mac OS X version 10.3, calling `AEResetTimer` did not reset the timeout value.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEInteraction.h`

AEResolve

Resolves an object specifier.

```
OSErr AEResolve (
   const AEDesc *objectSpecifier,
   short callbackFlags,
   AEDesc *theToken
);
```

Parameters

objectSpecifier

A pointer to the object specifier to resolve. See `AEDesc` (page 546).

callbackFlags

A value that determines what additional assistance, if any, your application can give the Apple Event Manager when it parses the object specifier. The value is specified by adding the desired constants described in "Callback Constants for the AEResolve Function" (page 571). Most applications use `kAEIDoMinimum`.

theToken

A pointer to a descriptor. On return, a token that identifies the Apple event objects specified by the `objectSpecifier` parameter. (Token is defined in `AEDisposeToken` (page 425).)

Your object accessor functions may need to create many tokens to resolve a single object specifier; this parameter contains only the final token that identifies the requested Apple event object.

Whenever the `AEResolve` function returns final token to your event handler as the result of the resolving an object specifier passed to `AEResolve`, your application must deallocate the memory used by the token. If your application uses complex tokens, it must dispose of the token by calling `AEDisposeToken` (page 425). If your application uses simple tokens, you can use either `AEDisposeToken` (page 425) or `AEDisposeDesc` (page 424). See `AEDesc` (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). The `AEResolve` function returns any result code returned by one of your application's object accessor functions or object callback functions. For example, an object accessor function can return `errAENoSuchObject` (–1728) when it can't find an Apple event object, or it can return more specific result codes. If any object accessor function or object callback function returns a result code other than `noErr` or `errAEEventNotHandled`, `AEResolve` immediately disposes of any existing tokens and returns. The result code it returns in this case is the result code returned by the object accessor function or the object callback function.

Discussion

If an Apple event parameter consists of an object specifier, your handler for the event typically calls the `AEResolve` function to begin the process of resolving the object specifier.

The `AEResolve` function resolves the object specifier passed in the `objectSpecifier` parameter with the help of your object accessor functions, described in "Object Accessor Callbacks" (page 523), and the object callback functions, described in "Object Callback Functions" (page 524).

For information on how to receive error information from the `AEResolve` function, see `OSLGetErrDescProcPtr` (page 541).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEObjects.h`

AEResumeTheCurrentEvent

Informs the Apple Event Manager that your application wants to resume the handling of a previously suspended Apple event or that it has completed the handling of the Apple event.

```
OSErr AEResumeTheCurrentEvent (
   const AppleEvent *theAppleEvent,
   const AppleEvent *reply,
   AEEventHandlerUPP dispatcher,
   SRefCon handlerRefcon
);
```

Parameters

theAppleEvent

A pointer to the Apple event to resume handling for. See `AppleEvent` (page 559).

reply

A pointer to the default reply provided by the Apple Event Manager for the Apple event. See `AppleEvent` (page 559).

dispatcher

One of the following:

- a universal procedural pointer to a function that the Apple Event Manager calls to handle the resumed event, or
- the constant `kAEUseStandardDispatch`, which tells the Apple Event Manager to handle the resumed event with its standard dispatching mechanism, or
- the constant `kAENoDispatch`, which tells the Apple Event Manager the Apple event has been completely processed and doesn't need to be dispatched.

See the `handlerRefcon` parameter for more information.

The dispatch constants are described in "Resume Event Dispatch Constants" (page 603).

See `AEEventHandlerUPP` (page 555).

handlerRefcon

If the `dispatcher` parameter specifies a universal procedure pointer to your routine, the reference constant is passed to your handler. If you pass the value `kAEUseStandardDispatch` or `kAENoDispatch` for the `dispatcher` parameter, you must pass 0 for the `handlerRefcon` parameter.

If the value of `dispatcher` is `kAEUseStandardDispatch`, the Apple Event Manager ignores the `handlerRefcon` parameter and instead passes the reference constant stored in the Apple event dispatch table entry for the resumed Apple event.

If the value of `dispatcher` is any other value then it is a universal procedure pointer to an event handler, and the Apple Event Manager passes the value from the `handlerRefcon` parameter as the reference constant when it calls the handler.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). This is the error result from the Apple event handler (or `errAEHandlerNotFound`). In most cases your application should ignore this error because it will be seen by the Apple event sender as the `keyErrorNumber` parameter in the reply.

Discussion

Applications call `AESuspendTheCurrentEvent` (page 497) to suspend handling of an Apple event and `AEResumeTheCurrentEvent` to resume it again. You typically call the `AESuspendTheCurrentEvent` function when your application needs to do some lengthy processing before responding to the event.

When your application calls the `AEResumeTheCurrentEvent` function, the Apple Event Manager resumes handling the specified Apple event using the handler specified in the `dispatcher` parameter, if any. If `kAENoDispatch` is specified in the `dispatcher` parameter, `AEResumeTheCurrentEvent` simply informs the Apple Event Manager that the specified event has been handled.

Special Considerations

This function is not thread-safe and, along with `AESuspendTheCurrentEvent`, should be called only on the main thread.

When your application suspends an Apple event, it does not need to dispose of the Apple event or the reply Apple event passed to the handler that suspends the event, whether or not the application eventually resumes the event. However, if the application will later resume the event, the handler that suspends the event should save a copy of the underlying data storage for the Apple event and the reply event. When resuming the event, you pass those copies to `AEResumeTheCurrentEvent`, which uses the information they contain to identify the original event and reply. For related information, see `AESuspendTheCurrentEvent` (page 497).

Make sure all processing involving the event or the reply has been completed before your application calls `AEResumeTheCurrentEvent`. Do not call `AEResumeTheCurrentEvent` for an event that was not suspended.

An Apple event handler that suspends an event should not immediately call `AEResumeTheCurrentEvent`, because the handler will generate an error. Instead, the handler should just return after suspending the event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AESend

Sends the specified Apple event.

```
OSErr AESend (
   const AppleEvent *theAppleEvent,
   AppleEvent *reply,
   AESendMode sendMode,
   AESendPriority sendPriority,
   SInt32 timeOutInTicks,
   AEIdleUPP idleProc,
   AEFilterUPP filterProc
);
```

Parameters

theAppleEvent
A pointer to the Apple event to send. See `AppleEvent` (page 559).

reply
A pointer to a reply Apple event. On return, contains the reply Apple event from the server application, if you specified the `kAEWaitReply` flag in the `sendMode` parameter. If you specify the `kAEQueueReply` flag in the `sendMode` parameter, you receive the reply Apple event in your event queue. If you specify `kAENoReply` flag, the reply Apple event is a null descriptor (one with descriptor type `typeNull`). If you specify `kAEWaitReply` in the `sendMode` parameter, and if the function returns successfully (see function result below), your application is responsible for using the `AEDisposeDesc` (page 424) function to dispose of the descriptor returned in the `reply` parameter.

sendMode
Specifies various options for how the server application should handle the Apple event. To obtain a value for this parameter, you add together constants to set bits that specify the reply mode, the interaction level, the application switch mode, the reconnection mode, and the return receipt mode. For more information, see "`AESendMode`" (page 566).

sendPriority
See the Version Notes section below for important information. A value that specifies the priority for processing the Apple event. You can specify normal or high priority, using the constants described in "`AESendMode`" (page 566). See `AESendPriority` (page 558).

timeOutInTicks

If the reply mode specified in the `sendMode` parameter is `kAEWaitReply`, or if a return receipt is requested, this parameter specifies the length of time (in ticks) that the client application is willing to wait for the reply or return receipt from the server application before timing out. Most applications should use the `kAEDefaultTimeout` constant, which tells the Apple Event Manager to provide an appropriate timeout duration. If the value of this parameter is `kNoTimeOut`, the Apple event never times out. These constants are described in "Timeout Constants" (page 605).

idleProc

A universal procedure pointer to a function that handles events (such as update, operating-system, activate, and null events) that your application receives while waiting for a reply. Your idle function can also perform other tasks (such as displaying a wristwatch or spinning beach ball cursor) while waiting for a reply or a return receipt.

If your application specifies the `kAEWaitReply` flag in the `sendMode` parameter and you wish your application to get periodic time while waiting for the reply to return, you must provide an idle function. Otherwise, you can pass a value of `NULL` for this parameter. For more information on the idle function, see `AEIdleProcPtr` (page 531).

filterProc

A universal procedure pointer to a function that determines which incoming Apple events should be received while the handler waits for a reply or a return receipt. If your application doesn't need to filter Apple events, you can pass a value of `NULL` for this parameter. If you do so, no application-oriented Apple events are processed while waiting. For more information on the filter function, see `AEFilterProcPtr` (page 530).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). The `AESend` function returns `noErr` if the Event Manager successfully sends the Apple event—this value does not indicate that the Apple event was handled successfully. If the handler returns a result code other than `noErr`, and if the client is waiting for a reply, `AESend` returns the result code in the `keyErrorNumber` parameter of the reply Apple event. For a result code other than `noErr`, you should not call the `AEDisposeDesc` (page 424) function to dispose of the descriptor returned in the `reply` parameter, because the descriptor is invalid.

Discussion

You typically create an Apple event to send with the `AECreateAppleEvent` (page 416) function and add information to it with the functions described in "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 398).

If the Apple Event Manager cannot find a handler for the Apple event in the server application's dispatch table or in the system dispatch table, it returns the result code `errAEEventNotHandled` to the server application (as the result of the `AEProcessAppleEvent` (page 457) function). If the client application is waiting for a reply, the Apple Event Manager also returns this result code to the client in the `keyErrorNumber` parameter of the reply event.

In addition to specifying the wait duration for replies, the `timeOutInTicks` parameter is used as a wait value when queuing events for other applications. The Apple Event Manager waits for the specified duration as it attempts to queue the event. If you specify `kAEWaitReply` and the target application quits or crashes after the event is queued but before the reply is returned, the Apple Event Manager returns a `sessionClosedErr` result code.

In some situations, there are advantages to sending Apple events with the `AESendMessage` (page 478) function. That function requires less overhead than `AESend` and it allows you to send Apple events without linking to the entire Carbon framework (and window server), as required by `AESend`. For more information on sending Apple events, see "Sending an Apple Event" in Apple Events Programming Guide.

Version Notes
In Mac OS 9 and earlier, you use the `sendMode` parameter to specify how the server should handle the Apple event. "`AESendMode`" (page 566) provides a complete description of the constants you use with this parameter. The `sendPriority` parameter is deprecated in Mac OS X and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AESendMessage

Sends an AppleEvent to a target process without some of the overhead required by `AESend`.

```
OSStatus AESendMessage (
   const AppleEvent *event,
   AppleEvent *reply,
   AESendMode sendMode,
   long timeOutInTicks
);
```

Parameters

event

A pointer to the Apple event to send.

reply

A pointer to a reply Apple event. On return, contains the reply Apple event from the server application, if you specified the `kAEWaitReply` flag in the `sendMode` parameter. If you specify the `kAEQueueReply` flag in the `sendMode` parameter, you receive the reply Apple event in your event queue. If you specify `kAENoReply` flag, the reply Apple event is a null descriptor (one with descriptor type `typeNull`). If you specify `kAEWaitReply` in the `sendMode` parameter, and if the function returns successfully (see function result below), your application is responsible for using the `AEDisposeDesc` (page 424) function to dispose of the descriptor returned in the `reply` parameter.

sendMode

Specifies various options for how the server application should handle the Apple event. To obtain a value for this parameter, you add together constants to set bits that specify the reply mode, the interaction level, the application switch mode, the reconnection mode, and the return receipt mode. For more information, see "`AESendMode`" (page 566).

timeOutInTicks

If the reply mode specified in the `sendMode` parameter is `kAEWaitReply`, or if a return receipt is requested, this parameter specifies the length of time (in ticks) that the client application is willing to wait for the reply or return receipt from the server application before timing out. Most applications should use the `kAEDefaultTimeout` constant, which tells the Apple Event Manager to provide an appropriate timeout duration. If the value of this parameter is `kNoTimeOut`, the Apple event never times out. These constants are described in "Timeout Constants" (page 605).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
The `AESendMessage` function allows you to send Apple events without linking to the entire Carbon framework, as required by `AESend` (page 476). Linking with Carbon brings in the HIToolbox framework, which requires that your application have a connection to the window server. Daemons and other applications that have

no interface but wish to send and receive Apple events can use the following functions for working with Apple events at a lower level: `AESendMessage`, `AEGetRegisteredMachPort` (page 445), `AEDecodeMessage` (page 421), and `AEProcessMessage` (page 458). See the descriptions for those functions for more information on when you might use them.

If the target of an event sent with `AESendMessage` is the current process (as specified by using `typeProcessSerialNumber` of `{ 0, kCurrentProcess }` in the Apple event being sent), the Apple event is dispatched directly to the appropriate event handler in your process and not serialized.

Special Considerations

The `AESendMessage` function is both asynchronous and thread-safe, so you could, for example, set up a thread to send an Apple event and wait for a reply. If you use threads, you must add a `typeReplyPortAttr` attribute to your event that identifies the Mach port on which to receive the reply.

However, due to a bug that was present prior to Mac OS X version 10.5, you could not safely dispose of a Mach port you created to use as the reply port. Disposing of the port could, rarely, lead to a crash, while failing to dispose of if leaked resources. The sample code project *AESendThreadSafe* shows how to safely work around the bug in earlier Mac OS versions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEMach.h`

AESetInteractionAllowed

Specifies user interaction preferences for responding to an Apple event when your application is the server application.

```
OSErr AESetInteractionAllowed (
   AEInteractAllowed level
);
```

Parameters

level

The desired user interaction level. Pass one of the values described in "User Interaction Level Constants" (page 605).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
If you don't set the user interaction level by calling `AESetInteractionAllowed`, the default level is `kAEInteractWithLocal` (which indicates that your server application may interact with the user in response to an Apple event only if the client application is on the same computer as the server application).

For additional information on interaction level, see `AESend` (page 476) and "`AESendMode`" (page 566).

See also `AESetInteractionAllowed` (page 479) and `AEInteractWithUser` (page 453).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEInteraction.h

AESetObjectCallbacks

Specifies the object callback functions for your application.

```
OSErr AESetObjectCallbacks (
   OSLCompareUPP myCompareProc,
   OSLCountUPP myCountProc,
   OSLDisposeTokenUPP myDisposeTokenProc,
   OSLGetMarkTokenUPP myGetMarkTokenProc,
   OSLMarkUPP myMarkProc,
   OSLAdjustMarksUPP myAdjustMarksProc,
   OSLGetErrDescUPP myGetErrDescProcPtr
);
```

Parameters

myCompareProc
Either a universal procedure pointer to the object comparison function provided by your application or NULL if no function is provided. See OSLCompareUPP (page 561).

myCountProc
Either a universal procedure pointer to the object-counting function provided by your application or NULL if no function is provided. See OSLCountUPP (page 561).

myDisposeTokenProc
Either a universal procedure pointer to the token disposal function provided by your application or NULL if no function is provided. (Token is defined in AEDisposeToken (page 425). See OSLDisposeTokenUPP (page 561).

myGetMarkTokenProc
Either a universal procedure pointer to the function for returning a mark token provided by your application or NULL if no function is provided. See OSLGetMarkTokenUPP (page 562).

myMarkProc
Either a universal procedure pointer to the object-marking function provided by your application or NULL if no function is provided. See OSLMarkUPP (page 562).

myAdjustMarksProc
Either a universal procedure pointer to the mark-adjusting function provided by your application or NULL if no function is provided. See OSLAdjustMarksUPP (page 561).

myGetErrDescProcPtr
Either a universal procedure pointer to the error callback function provided by your application or NULL if no function is provided. See OSLGetErrDescUPP (page 562).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This function is just a convenient wrapper for AEInstallSpecialHandler (page 452). You can manipulate the special handler table with more control using the routines described in "Managing Special Handler Dispatch Tables" (page 404).

Your application can provide only one each of the object callback functions specified by AESetObjectCallbacks—one object comparison function, one object-counting function, and so on. As a result, each of these callback functions must perform the requested task (comparing, counting, and so on)

for all the object classes that your application supports. In contrast, your application may provide many different object accessor functions if necessary, depending on the object classes and token types your application supports. You install object accessor functions with `AEInstallObjectAccessor` (page 451).

To replace object callback functions that have been previously installed, you can call `AESetObjectCallbacks` again. Each additional call to `AESetObjectCallbacks` replaces any object callback functions installed by previous calls. Only those functions you specify are replaced; to avoid replacing existing callback functions, specify a value of `NULL` for the functions you don't want to replace.

You cannot use `AESetObjectCallbacks` to replace system object callback functions or object accessor functions. To install system object callback functions, use the function `AEInstallSpecialHandler` (page 452).

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AESetTheCurrentEvent

Specifies a current Apple event to take the place of the one your application has suspended.

```
OSErr AESetTheCurrentEvent (
   const AppleEvent *theAppleEvent
);
```

Parameters

theAppleEvent
A pointer to the Apple event to handle as the current event. See `AppleEvent` (page 559).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
There is usually no reason for your application to use the `AESetTheCurrentEvent` function. Instead of calling this function, your application should let the Apple Event Manager set the current Apple event through its standard dispatch mechanism.

If you need to avoid the dispatch mechanism, you must use the `AESetTheCurrentEvent` function only in the following way:

1. Your application suspends handling of an Apple event by calling the `AESuspendTheCurrentEvent` (page 497) function.

2. Your application calls the `AESetTheCurrentEvent` function. This informs the Apple Event Manager that your application is handling the suspended Apple event. In this way, any functions that call the `AEGetTheCurrentEvent` (page 447) function can ascertain which event is currently being handled.

3. When your application finishes handling the Apple event, it calls the `AEResumeTheCurrentEvent` (page 474) function with the value `kAENoDispatch` to tell the Apple Event Manager that the event has been processed and need not be dispatched.

Special Considerations

This function is not thread-safe and should only be called on the main thread.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AESizeOfAttribute

Gets the size and descriptor type of an Apple event attribute from a descriptor of type `AppleEvent`.

```
OSErr AESizeOfAttribute (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType *typeCode,
   Size *dataSize
);
```

Parameters

theAppleEvent
A pointer to the Apple event to get the attribute data from. See `AppleEvent` (page 559).

theAEKeyword
The keyword that specifies the attribute. Some keyword constants are described in "Keyword Attribute Constants" (page 593). See `AEKeyword` (page 556).

typeCode
A pointer to a descriptor type. On return, specifies the descriptor type of the attribute. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). Can be `NULL`. See `DescType` (page 560).

dataSize
A pointer to a size variable. On return, the length, in bytes, of the data in the attribute. Can be `NULL`.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESizeOfFlattenedDesc

Returns the amount of buffer space needed to store the descriptor after flattening it.

```
Size AESizeOfFlattenedDesc (
   const AEDesc *theAEDesc
);
```

Parameters

theAEDesc

A pointer to the descriptor to be flattened. See `AEDesc` (page 546).

Return Value

The size, in bytes, required to store the flattened descriptor.

Discussion

You call this function before calling `AEFlattenDesc` (page 426) to determine the required size of the buffer for the flatten operation.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AESizeOfKeyDesc

Gets the size and descriptor type of an Apple event parameter from a descriptor of type `AERecord`.

```
OSErr AESizeOfKeyDesc (
    const AppleEvent *theAERecord,
    AEKeyword theAEKeyword,
    DescType *typeCode,
    Size *dataSize
);
```

Parameters

theAERecord

A pointer to the Apple event record to get the parameter data from.

theAEKeyword

The keyword that specifies the desired parameter. Some keyword parameter constants are described in "Keyword Parameter Constants" (page 595). See `AEKeyword` (page 556).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataSize

A pointer to a size variable. On return, the length, in bytes, of the data in the Apple event parameter.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This function is declared as a macro that invokes `AESizeOfParam` (page 485), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See `AESizeOfParam` (page 485).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESizeOfNthItem

Gets the data size and descriptor type of the descriptor at a specified position in a descriptor list.

```
OSErr AESizeOfNthItem (
   const AEDescList *theAEDescList,
   long index,
   DescType *typeCode,
   Size *dataSize
);
```

Parameters

theAEDescList
A pointer to the descriptor list containing the descriptor. See `AEDescList` (page 553).

index
A one-based positive integer indicating the position of the descriptor to get the data size for. `AESizeOfNthItem` returns an error if you pass zero, a negative number, or a value that is out of range.

typeCode
A pointer to a descriptor type. On return, specifies the descriptor type of the descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataSize
A pointer to a size variable. On return, the length (in bytes) of the data in the descriptor.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESizeOfParam

Gets the size and descriptor type of an Apple event parameter from a descriptor of type `AERecord` or `AppleEvent`.

```
OSErr AESizeOfParam (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType *typeCode,
   Size *dataSize
);
```

Parameters

theAppleEvent

A pointer to the Apple event to get the parameter data from. See `AppleEvent` (page 559).

theAEKeyword

The keyword that specifies the desired parameter. Some keyword parameter constants are described in "Keyword Parameter Constants" (page 595). See `AEKeyword` (page 556).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataSize

A pointer to a size variable. On return, the length, in bytes, of the data in the Apple event parameter.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AEStreamClose

Closes and deallocates an `AEStreamRef`.

```
OSStatus AEStreamClose (
   AEStreamRef ref,
   AEDesc *desc
);
```

Parameters

ref

An `AEStreamRef` (page 558)containing the stream data.

desc

A pointer to a descriptor for receiving a the stream data, or `NULL` if you want to discard the data. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Use this function to dispose of an `AEStreamRef` you created using `AEStreamCreateEvent` (page 487), `AEStreamOpen` (page 489), or `AEStreamOpenEvent` (page 490). To retrieve the resulting descriptor from the stream prior to disposal, pass in a pointer to an `AEDesc` structure in the *`desc`* parameter. If this parameter exists, `AEStreamClose` fills in the descriptor with the stream data. If the stream contains invalid information, possibly due to improperly balanced calls to "AEStream" functions, the returned descriptor type is set to `typeNull`.

Regardless of any errors returned by this function, it is always safe to call `AEDisposeDesc` (page 424) on the returned descriptor.

Specifying `NULL` for the *`desc`* parameter causes `AEStreamClose` to discard the stream data and dispose of the `AEStreamRef`. When you call `AEStreamClose` in this way, you do not need to worry about balancing nested calls to "AEStream" functions. This technique is particularly useful during error-handling situations where you need to dispose of a stream but do not know its exact state.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCloseDesc

Marks the end of a descriptor in an `AEStreamRef`.

```
OSStatus AEStreamCloseDesc (
   AEStreamRef ref
);
```

Parameters

`ref`
An `AEStreamRef` (page 558)containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Call this function to balance a preceding call to `AEStreamOpenDesc` (page 489) or `AEStreamOpenKeyDesc` (page 490). This function completes the definition of the `AEDesc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCloseList

Marks the end of a list of descriptors in an `AEStreamRef`.

```
OSStatus AEStreamCloseList (
   AEStreamRef ref
);
```

Parameters

ref

An `AEStreamRef` (page 558)containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Call this function to balance a preceding call to `AEStreamOpenList` (page 491). This function completes the definition of the `AEDescList`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCloseRecord

Marks the end of a record in an `AEStreamRef`.

```
OSStatus AEStreamCloseRecord (
   AEStreamRef ref
);
```

Parameters

ref

An `AEStreamRef` (page 558)containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Call this function to balance a preceding call to `AEStreamOpenRecord` (page 491). This function completes the definition of the Apple event record.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCreateEvent

Creates a new Apple event and opens a stream for writing data to it.

```
AEStreamRef AEStreamCreateEvent (
   AEEventClass clazz,
   AEEventID id,
   DescType targetType,
   const void *targetData,
   Size targetLength,
   SInt16 returnID,
   SInt32 transactionID
);
```

Parameters

clazz

The event class of the Apple event. See `AEEventClass` (page 555).

id

The event ID of the Apple event. See `AEEventID` (page 556).

targetType

The address type for the addressing information in the next two parameters. Usually contains one of the following values: `typeApplSignature`. `typeKernelProcessID`, or `typeProcessSerialNumber`. See `DescType` (page 560).

targetData

A pointer to the address information. The data in this pointer must match the data associated with the specified *targetType*.

targetLength

The number of bytes pointed to by the `targetData` parameter.

returnID

The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID. The return ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 589). See `AEReturnID` (page 558).

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events. This transaction ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 589). See `AETransactionID` (page 559).

Return Value

An `AEStreamRef` (page 558) associated with the new event.

Discussion

This routine effectively combines a call to `AECreateAppleEvent` (page 416) followed by a call to `AEStreamOpenEvent` (page 490) to create a new Apple event in the stream. You can use the returned `AEStreamRef` to add parameters to the new Apple event.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEHelpers.h`

AEStreamOpen

Opens a new `AEStreamRef` for use in building a descriptor.

```
AEStreamRef AEStreamOpen (
   void
);
```

Return Value
A new `AEStreamRef` (page 558) or `NULL` if the stream data structures cannot be allocated.

Discussion
This function creates a new stream for use in describing the contents of a descriptor, descriptor list, or Apple event record (`AEDesc`, `AEDescList`, or `AERecord`).

You can use the returned `AEStreamRef` with other "AEStream" routines to build the contents of a descriptor. When you are done building the descriptor, use `AEStreamClose` (page 485) to close the stream.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenDesc

Marks the beginning of a descriptor in an `AEStreamRef`.

```
OSStatus AEStreamOpenDesc (
   AEStreamRef ref,
   DescType newType
);
```

Parameters

ref
An `AEStreamRef` (page 558) containing the stream data.

newType
A type code for the new `AEDesc` being added to the stream. See `DescType` (page 560).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Use this routine to mark the beginning of a descriptor definition in an `AEDesc`. After calling this routine, you should call `AEStreamWriteData` (page 494) one or more times to write the descriptor data to the stream. When you are done writing data, you must call `AEStreamCloseDesc` (page 486) to complete the descriptor definition.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenEvent

Opens a stream for an existing Apple event.

```
AEStreamRef AEStreamOpenEvent (
   AppleEvent *event
);
```

Parameters

event

An existing Apple event. See `AppleEvent` (page 559).

Return Value

An `AEStreamRef` (page 558) for the Apple event or `NULL` if the stream data structures could not be allocated.

Discussion

Use this function to open a stream and add parameters to an existing Apple event. This function copies any parameters already in the Apple event to the stream prior to returning the `AEStreamRef`. When you are done adding parameters, use `AEStreamClose` (page 485) to save them to the Apple event and close the stream.

If there is not enough available storage to complete the operation, `AEStreamOpenEvent` returns `NULL` and leaves the Apple event unchanged.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEHelpers.h`

AEStreamOpenKeyDesc

Marks the beginning of a key descriptor in an `AEStreamRef`.

```
OSStatus AEStreamOpenKeyDesc (
   AEStreamRef ref,
   AEKeyword key,
   DescType newType
);
```

Parameters

ref

An `AEStreamRef` (page 558) containing the stream data.

key

The `AEKeyword` associated with the new descriptor being added to the stream. See `AEKeyword` (page 556).

newType

A type code for the new `AEDesc` being added to the stream. See `DescType` (page 560).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Use this routine to mark the beginning of a keyword/descriptor definition in an Apple event record. After calling this routine, you should call `AEStreamWriteData` (page 494) one or more times to write the record data to the stream. When you are done writing data, you must call `AEStreamCloseDesc` (page 486) to complete the record definition.

This routine must be called only as part of an Apple event record definition. You cannot use this routine to write keyword/descriptor definitions to other descriptor types, such as an `AEDesc` or `AEDescList`, even if those types are nested inside an Apple event record. In situations where you need to create nested records, this routine opens a new keyword/descriptor definition in the Apple event record associated with the most recent call to `AEStreamOpenRecord` (page 491).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenList

Marks the beginning of a descriptor list in an `AEStreamRef`.

```
OSStatus AEStreamOpenList (
   AEStreamRef ref
);
```

Parameters
ref
An `AEStreamRef` (page 558) containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This routine marks the beginning of a sequence of zero or more descriptor definitions that you use to build an `AEDescList` structure. After calling this routine, you can write any number of `AEDesc`, `AEDescList`, or `AERecord` structures to the stream as elements of the list. When you are done, you must call `AEStreamCloseList` (page 486) to complete the `AEDescList` definition.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenRecord

Marks the beginning of an Apple event record in an `AEStreamRef`.

```
OSStatus AEStreamOpenRecord (
   AEStreamRef ref,
   DescType newType
);
```

Parameters

ref

An `AEStreamRef` (page 558) containing the stream data.

newType

A type code for the new record you are adding to the stream. This value can be `typeAERecord` or any other appropriate value. See `DescType` (page 560).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
This routine marks the beginning of a sequence of zero or more keyword/descriptor definitions that you use to build an `AERecord` structure. You must balance each call to this method with a corresponding call to `AEStreamCloseRecord` (page 487).

For information on adding keyword/descriptor data to the record, see the `AEStreamOpenKeyDesc` (page 490), `AEStreamWriteKey` (page 495), and `AEStreamWriteKeyDesc` (page 496) routines.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOptionalParam

Designates a parameter in an Apple event as optional.

```
OSStatus AEStreamOptionalParam (
   AEStreamRef ref,
   AEKeyword key
);
```

Parameters

ref

An `AEStreamRef` (page 558) containing the stream data.

key

The `AEKeyword` associated with any keyword/descriptor pair in an Apple event. See `AEKeyword` (page 556).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Calls to this routine must be preceded by a call to either `AEStreamCreateEvent` (page 487) or `AEStreamOpenEvent` (page 490).

The descriptor associated with the specified *key* does not need to exist before you call this routine.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEHelpers.h

AEStreamSetRecordType

Sets the type of the most recently created record in an AEStreamRef.

```
OSStatus AEStreamSetRecordType (
   AEStreamRef ref,
   DescType newType
);
```

Parameters

ref
An AEStreamRef (page 558) containing the stream data.

newType
The new type code for the AERecord being added to the stream. See DescType (page 560).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Use this routine to change the type of a record after it has been opened. You must call this routine between calls to AEStreamOpenRecord (page 491) and AEStreamCloseRecord (page 487). The type you specify in the *newType* parameter replaces the previous type specified by AEStreamOpenRecord (page 491).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEHelpers.h

AEStreamWriteAEDesc

Copies an existing descriptor into an AEStreamRef.

```
OSStatus AEStreamWriteAEDesc (
   AEStreamRef ref,
   const AEDesc *desc
);
```

Parameters

ref
An AEStreamRef (page 558) containing the stream data.

desc
A pointer to the descriptor you want to copy into the stream. See AEDesc (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
You can use this routine to incorporate an existing descriptor into the stream. For example, you could use this routine if you had a complex descriptor you wanted to add to multiple streams, but which would be costly to create each time.

Do not use `AEStreamOpenDesc` (page 489) and `AEStreamCloseDesc` (page 486) with this routine to open and close the descriptor.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteData

Appends data to the current descriptor in an `AEStreamRef`.

```
OSStatus AEStreamWriteData (
   AEStreamRef ref,
   const void *data,
   Size length
);
```

Parameters

ref
An `AEStreamRef` (page 558) containing the stream data.

data
A pointer to the block of memory containing the descriptor data. This routine copies the memory block immediately, so you do not need to retain it for the benefit of this routine.

length
The number of bytes pointed to by the *data* parameter.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
You can call this routine any number of times to build up the data contents of the descriptor incrementally. You must precede calls to this routine by a call to either `AEStreamOpenDesc` (page 489) or `AEStreamOpenKeyDesc` (page 490). When you are done adding data to the descriptor, call `AEStreamCloseDesc` (page 486) to complete the descriptor definition.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteDesc

Appends the data for a complete descriptor to an `AEStreamRef`.

```
OSStatus AEStreamWriteDesc (
   AEStreamRef ref,
   DescType newType,
   const void *data,
   Size length
);
```

Parameters

ref

An `AEStreamRef` (page 558) containing the stream data.

newType

A type code for the new `AEDesc` being added to the stream. See `DescType` (page 560).

data

A pointer to the block of memory containing the descriptor data. This routine copies the memory block immediately, so you do not need to retain it for the benefit of this routine.

length

The number of bytes pointed to by the *data* parameter.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Use this routine to write the data for a descriptor to the stream. When using this routine, you must supply all of the descriptor data at once.

Do not use `AEStreamOpenDesc` (page 489) and `AEStreamCloseDesc` (page 486) with this routine to open and close the descriptor.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteKey

Marks the beginning of a keyword/descriptor pair for a descriptor in an `AEStreamRef`.

```
OSStatus AEStreamWriteKey (
   AEStreamRef ref,
   AEKeyword key
);
```

Parameters

ref

An `AEStreamRef` (page 558) containing the stream data.

key

The `AEKeyword` associated with the new descriptor being added to the stream. See `AEKeyword` (page 556).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
You must follow this call with a sequence of "AEStream" calls to specify exactly one descriptor that goes with the keyword. The descriptor you create can be of type `AEDesc`, `AEDescList`, or `AERecord`.

If you are creating nested descriptors, this routine begins a new keyword/descriptor pair for the descriptor most recently opened by a call to `AEStreamWriteKey` (page 495) or `AEStreamOpenEvent` (page 490). You cannot use this routine to write parameters to any other types of descriptors, even if they are nested inside of an `AERecord`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteKeyDesc

Writes a complete keyword/descriptor pair to an `AEStreamRef`.

```
OSStatus AEStreamWriteKeyDesc (
   AEStreamRef ref,
   AEKeyword key,
   DescType newType,
   const void *data,
   Size length
);
```

Parameters

ref
An `AEStreamRef` (page 558) containing the stream data.

key
The `AEKeyword` associated with the new descriptor being added to the stream. See `AEKeyword` (page 556).

newType
A type code for the new `AEDesc` being added to the stream. See `DescType` (page 560).

data
A pointer to the block of memory containing the descriptor data. This routine copies the memory block immediately, so you do not need to retain it for the benefit of this routine.

length
The number of bytes pointed to by the *data* parameter.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Use this routine to add a descriptor to the currently open `AERecord` inside a stream. You cannot use this routine to write parameters to any other types of descriptors, even if they are nested inside of an `AERecord`. This routine can only be called in between calls to `AEStreamOpenRecord` (page 491) and `AEStreamCloseRecord` (page 487).

This method is analogous to the Apple Event Manager routine `AEPutParamPtr` (page 464), except it is for use with streams.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEHelpers.h

AESuspendTheCurrentEvent

Suspends the processing of the Apple event that is currently being handled.

```
OSErr AESuspendTheCurrentEvent (
   const AppleEvent *theAppleEvent
);
```

Parameters

theAppleEvent

A pointer to the Apple event to suspend handling for. If the pointed-to Apple event is not the current event, AESuspendTheCurrentEvent does nothing and returns noErr. See AppleEvent (page 559).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
You typically call the AESuspendTheCurrentEvent function from an Apple event handler function, such as when your application needs to do some lengthy processing before responding to the event. After a successful call to this function, you are not required to return a result or a reply for the Apple event that was being handled. You can, however, return a result if you later call the AEResumeTheCurrentEvent (page 474) function to resume event processing.

Whether you will resume the suspended Apple event or not, you do not need to dispose of the Apple event or the reply Apple event passed to your handler. However, if your handler will later resume the event, you must save a copy of the underlying data storage for the Apple event and the reply event. When resuming the event, you pass those copies to AEResumeTheCurrentEvent (page 474), which uses the information they contain to identify the original event and reply.

You cannot merely save the pointers that are passed to your handler because they do not persist after your handler returns (although the underlying Apple events do persist). Use a function such as AEDuplicateDesc (page 426) to obtain copies of the Apple event and reply event. Later, after calling AEResumeTheCurrentEvent to resume the event, call AEDisposeDesc (page 424) to dispose of the copies.

Special Considerations

This function is not thread-safe and, along with AEResumeTheCurrentEvent, should be called only on the main thread.

If your application suspends handling of an Apple event it sends to itself, the Apple Event Manager immediately returns from the AESend (page 476) call with the error code errAETimeout, regardless of the parameters specified in the call to AESend. The function calling AESend should take the timeout error as confirmation that the event was sent.

As with other calls to AESend that return a timeout error, the handler continues to process the event nevertheless. The handler's reply, if any, is provided in the reply event when the handling is completed. The Apple Event Manager provides no notification that the reply is ready. If no data has yet been placed in the reply event, the Apple Event Manager returns errAEReplyNotArrived when your application attempts to extract data from the reply.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEInteraction.h

AEUnflattenDesc

Unflattens the data in the passed buffer and creates a descriptor from it.

```
OSStatus AEUnflattenDesc (
   const void *buffer,
   AEDesc *result
);
```

Parameters

buffer

A pointer to memory, allocated by the application, that contains flattened data produced by a previous call to AEFlattenDesc (page 426).

result

A null descriptor. On successful completion, points to a descriptor created from the flattened data. The caller is responsible for disposing of the descriptor.

Return Value
A result code. Returns paramErr if the flattened data in buffer is found to be invalid. See "Apple Event Manager Result Codes" (page 636) for other possible values.

Discussion
This function assumes the passed buffer contains valid flattened data, produced by a previous call to AEFlattenDesc (page 426). See that function for a description of when you might want to flatten and unflatten descriptors, and of possible limitations.

Flattening and unflattening works across OS versions, including between Mac OS 9 and Mac OS X.

Flattening is endian-neutral. That is, you can save flattened data on a machine that is either big-endian or little-endian, then retrieve and unflatten the data on either type of machine, without any special steps by your application.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

CreateCompDescriptor

Creates a comparison descriptor that specifies how to compare one or more Apple event objects with either another Apple event object or a descriptor.

```
OSErr CreateCompDescriptor (
   DescType comparisonOperator,
   AEDesc *operand1,
   AEDesc *operand2,
   Boolean disposeInputs,
   AEDesc *theDescriptor
);
```

Parameters

comparisonOperator

The comparison operator for comparing the descriptors in the `operand1` and `operand2` parameters. The standard comparison operators are defined in "Comparison Operator Constants" (page 574).

The actual comparison of the two operands is performed by the object comparison function provided by the client application. The way a comparison operator is interpreted is up to each application.

See `DescType` (page 560).

operand1

A pointer to an object specifier. See `AEDesc` (page 546).

operand2

A pointer to a descriptor (which can be an object specifier or any other descriptor) whose value is compared to the value of `operand1`. See `AEDesc` (page 546).

disposeInputs

A Boolean value. Pass `TRUE` if the function should automatically dispose of any descriptors you have provided in the `operand1` and `operand2` parameters to the function. Pass `FALSE` if your application will dispose of the descriptors itself. A value of `FALSE` may be more efficient for some applications because it allows them to reuse descriptors.

theDescriptor

A pointer to a descriptor. On successful return, the comparison descriptor created by `CreateCompDescriptor`. Your application must dispose of this descriptor after it has finished using it. See `AEDesc` (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEPackObject.h`

CreateLogicalDescriptor

Creates a logical descriptor that specifies a logical operator and one or more logical terms for the Apple Event Manager to evaluate.

```
OSErr CreateLogicalDescriptor (
   AEDescList *theLogicalTerms,
   DescType theLogicOperator,
   Boolean disposeInputs,
   AEDesc *theDescriptor
);
```

Parameters

theLogicalTerms

A pointer to a list containing comparison descriptors (typeLogicalDescriptor), logical descriptors (typeCompDescriptor), or both. If the value of the parameter theLogicOperator is kAEAND or kAEOR, the list can contain any number of descriptors. If the value of the parameter theLogicOperator is kAENOT, logically this list should contain a single descriptor. However, the function will not return an error if the list contains more than one descriptor for a logical operator of kAENOT. See AEDescList (page 553).

theLogicOperator

A logical operator represented by one of the constants described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 575). What you pass for this parameter helps determine what you pass for the theLogicalTerms parameter. See DescType (page 560).

disposeInputs

A Boolean value. Pass TRUE if the function should automatically dispose of the descriptors you have provided in the theLogicalTerms parameter or (FALSE) if your application will. A value of FALSE may be more efficient for some applications because it allows them to reuse descriptors.

theDescriptor

A pointer to a descriptor. On successful return, the logical descriptor created by CreateLogicalDescriptor. Your application must dispose of this descriptor after it has finished using it. See AEDesc (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion

The CreateLogicalDescriptor function creates a logical descriptor, which specifies a logical operator and one or more logical terms for the Apple Event Manager to evaluate.

Availability

Available in Mac OS X v10.0 and later.

Declared In

AEPackObject.h

CreateObjSpecifier

Assembles an object specifier that identifies one or more Apple event objects, from other descriptors.

```
OSErr CreateObjSpecifier (
   DescType desiredClass,
   AEDesc *theContainer,
   DescType keyForm,
   AEDesc *keyData,
   Boolean disposeInputs,
   AEDesc *objSpecifier
);
```

Parameters

desiredClass

The object class of the desired Apple event objects. See `DescType` (page 560).

theContainer

A pointer to a descriptor that describes the container for the requested object, usually in the form of another object specifier. See `AEDesc` (page 546).

keyForm

The key form for the object specifier.

keyData

A pointer to a descriptor that supplies the key data for the object specifier.

disposeInputs

A Boolean value. Pass (`TRUE`) if the function should dispose of the descriptors for the `theContainer` and `keyData` parameters or (`FALSE`) if your application will. A value of `FALSE` may be more efficient for some applications because it allows them to reuse descriptors.

objSpecifier

On successful return, a pointer to the object specifier created by the `CreateObjSpecifier` function. If the function returns successfully, your application should call the `AEDisposeDesc` (page 424) function to dispose of this descriptor after it has finished using it.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEPackObject.h`

CreateOffsetDescriptor

Creates an offset descriptor that specifies the position of an element in relation to the beginning or end of its container.

```
OSErr CreateOffsetDescriptor (
   long theOffset,
   AEDesc *theDescriptor
);
```

Parameters

theOffset

A positive integer that specifies the offset from the beginning of the container (the first element has an offset of 1), or a negative integer that specifies the offset from the end (the last element has an offset of –1).

theDescriptor

A pointer to a descriptor. On successful return, the offset descriptor created by `CreateOffsetDescriptor`. On error, returns a null descriptor. Your application must dispose of the descriptor after it has finished using it. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEPackObject.h`

CreateRangeDescriptor

Creates a range descriptor that specifies a series of consecutive elements in the same container.

```
OSErr CreateRangeDescriptor (
   AEDesc *rangeStart,
   AEDesc *rangeStop,
   Boolean disposeInputs,
   AEDesc *theDescriptor
);
```

Parameters

rangeStart

A pointer to an object specifier that identifies the first Apple event object in the range. See `AEDesc` (page 546).

rangeStop

A pointer to an object specifier that identifies the last Apple event object in the range. See `AEDesc` (page 546).

disposeInputs

A Boolean value. Pass (`TRUE`) if the function should dispose of the descriptors for the `rangeStart` and `rangeStop` parameters and set them to the null descriptor or (`FALSE`) if your application will. A value of `FALSE` may be more efficient for some applications because it allows them to reuse descriptors.

theDescriptor

A pointer to a descriptor. On successful return, the range descriptor created by `CreateRangeDescriptor`. Your application must dispose of this descriptor after it has finished using it. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Although the `rangeStart` and `rangeStop` parameters can be any object specifiers—including object specifiers that specify more than one Apple event object—most applications expect these parameters to specify single Apple event objects.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEPackObject.h`

DisposeAECoerceDescUPP

Disposes of a universal procedure pointer to a function that coerces data stored in a descriptor.

```
void DisposeAECoerceDescUPP (
   AECoerceDescUPP userUPP
);
```

Discussion
See the `AECoerceDescProcPtr` (page 524) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DisposeAECoercePtrUPP

Disposes of a universal procedure pointer to a function that coerces data stored in a buffer.

```
void DisposeAECoercePtrUPP (
   AECoercePtrUPP userUPP
);
```

Discussion
See the `AECoercePtrProcPtr` (page 525) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DisposeAEDisposeExternalUPP

Disposes of a universal procedure pointer to a function that disposes of data supplied to the `AECreateDescFromExternalPtr` function.

```
void DisposeAEDisposeExternalUPP (
   AEDisposeExternalUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to be disposed of. See `AEDisposeExternalUPP` (page 555).

Discussion
See the `AECreateDescFromExternalPtr` (page 418) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

DisposeAEEventHandlerUPP

Disposes of a universal procedure pointer to an event handler function.

```
void DisposeAEEventHandlerUPP (
   AEEventHandlerUPP userUPP
);
```

Discussion
See the `AEEventHandlerProcPtr` (page 528) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DisposeAEFilterUPP

Disposes of a universal procedure pointer to an Apple event filter function.

```
void DisposeAEFilterUPP (
   AEFilterUPP userUPP
);
```

Discussion
See the `AEFilterProcPtr` (page 530) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

DisposeAEIdleUPP

Disposes of a universal procedure pointer to an Apple event idle function.

```
void DisposeAEIdleUPP (
   AEIdleUPP userUPP
);
```

Discussion
See the `AEIdleProcPtr` (page 531) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

DisposeOSLAccessorUPP

Disposes of a universal procedure pointer to an object accessor function.

```
void DisposeOSLAccessorUPP (
   OSLAccessorUPP userUPP
);
```

Discussion
See the `OSLAccessorProcPtr` (page 533) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLAdjustMarksUPP

Disposes of a universal procedure pointer to an object callback adjust marks function.

```
void DisposeOSLAdjustMarksUPP (
   OSLAdjustMarksUPP userUPP
);
```

Discussion
See the `OSLAdjustMarksProcPtr` (page 535) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLCompareUPP

Disposes of a universal procedure pointer to an object callback comparison function.

```
void DisposeOSLCompareUPP (
   OSLCompareUPP userUPP
);
```

Discussion
See the `OSLCompareProcPtr` (page 536) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLCountUPP

Disposes of a universal procedure pointer to an object callback count function.

```
void DisposeOSLCountUPP (
   OSLCountUPP userUPP
);
```

Discussion
See the `OSLCountProcPtr` (page 538) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLDisposeTokenUPP

Disposes of a universal procedure pointer to an object callback dispose token function.

```
void DisposeOSLDisposeTokenUPP (
   OSLDisposeTokenUPP userUPP
);
```

Discussion
See the `OSLDisposeTokenProcPtr` (page 539) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLGetErrDescUPP

Disposes of a universal procedure pointer to an object callback get error descriptor function.

```
void DisposeOSLGetErrDescUPP (
   OSLGetErrDescUPP userUPP
);
```

Discussion
See the `OSLGetErrDescProcPtr` (page 541) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLGetMarkTokenUPP

Disposes of a universal procedure pointer to an object callback get mark function.

```
void DisposeOSLGetMarkTokenUPP (
   OSLGetMarkTokenUPP userUPP
);
```

Discussion
See the `OSLGetMarkTokenProcPtr` (page 542) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLMarkUPP

Disposes of a universal procedure pointer to an object callback mark function.

```
void DisposeOSLMarkUPP (
   OSLMarkUPP userUPP
);
```

Discussion
See the `OSLMarkProcPtr` (page 544) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeAECoerceDescUPP

Calls a universal procedure pointer to a function that coerces data stored in a descriptor.

```
OSErr InvokeAECoerceDescUPP (
   const AEDesc *fromDesc,
   DescType toType,
   SRefCon handlerRefcon,
   AEDesc *toDesc,
   AECoerceDescUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `AECoerceDescProcPtr` (page 524) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

InvokeAECoercePtrUPP

Calls a universal procedure pointer to a function that coerces data stored in a buffer.

```
OSErr InvokeAECoercePtrUPP (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   DescType toType,
   SRefCon handlerRefcon,
   AEDesc *result,
   AECoercePtrUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `AECoercePtrProcPtr` (page 525) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

InvokeAEDisposeExternalUPP

Calls a dispose external universal procedure pointer.

```
void InvokeAEDisposeExternalUPP (
   const void *dataPtr,
   Size dataLength,
   SRefCon refcon,
   AEDisposeExternalUPP userUPP
);
```

Parameters

dataPtr
A pointer to the data to be disposed of. The data must be immutable and must not be freed until this UPP is called.

dataLength
The length, in bytes, of the data to be disposed of.

refcon
A reference constant, supplied by your application, that you can use in your dispose function.

Discussion
See the `AEDisposeExternalProcPtr` (page 527) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

InvokeAEEventHandlerUPP

Calls an event handler universal procedure pointer.

```
OSErr InvokeAEEventHandlerUPP (
   const AppleEvent *theAppleEvent,
   AppleEvent *reply,
   SRefCon handlerRefcon,
   AEEventHandlerUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `AEEventHandlerProcPtr` (page 528) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

InvokeAEFilterUPP

Calls an Apple event filter universal procedure pointer.

```
Boolean InvokeAEFilterUPP (
   EventRecord *theEvent,
   SInt32 returnID,
   AETransactionID transactionID,
   const AEAddressDesc *sender,
   AEFilterUPP userUPP
);
```

Return Value
The return value of the callback function. The filter routine returns `TRUE` to accept the Apple event or `FALSE` to filter it out.

Discussion
See the `AEFilterProcPtr` (page 530) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

InvokeAEIdleUPP

Calls an Apple event idle universal procedure pointer.

```
Boolean InvokeAEIdleUPP (
   EventRecord *theEvent,
   SInt32 *sleepTime,
   RgnHandle *mouseRgn,
   AEIdleUPP userUPP
);
```

Return Value
The return value of the callback function. The filter routine returns `TRUE` if your application is no longer willing to wait for a reply from the server or for the user to bring the application to the front. It returns `FALSE` if your application is still willing to wait.

Discussion
See the `AEIdleProcPtr` (page 531) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

InvokeOSLAccessorUPP

Calls an object accessor universal procedure pointer.

```
OSErr InvokeOSLAccessorUPP (
   DescType desiredClass,
   const AEDesc *container,
   DescType containerClass,
   DescType form,
   const AEDesc *selectionData,
   AEDesc *value,
   SRefCon accessorRefcon,
   OSLAccessorUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLAccessorProcPtr` (page 533) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLAdjustMarksUPP

Calls an object callback adjust marks universal procedure pointer.

```
OSErr InvokeOSLAdjustMarksUPP (
   long newStart,
   long newStop,
   const AEDesc *markToken,
   OSLAdjustMarksUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLAdjustMarksProcPtr` (page 535) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLCompareUPP

Calls an object callback comparison universal procedure pointer.

```
OSErr InvokeOSLCompareUPP (
   DescType oper,
   const AEDesc *obj1,
   const AEDesc *obj2,
   Boolean *result,
   OSLCompareUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLCompareProcPtr` (page 536) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLCountUPP

Calls an object callback count universal procedure pointer.

```
OSErr InvokeOSLCountUPP (
   DescType desiredType,
   DescType containerClass,
   const AEDesc *container,
   long *result,
   OSLCountUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLCountProcPtr` (page 538) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLDisposeTokenUPP

Calls an object callback dispose token universal procedure pointer.

```
OSErr InvokeOSLDisposeTokenUPP (
   AEDesc *unneededToken,
   OSLDisposeTokenUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLDisposeTokenProcPtr` (page 539) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLGetErrDescUPP

Calls an object callback get error descriptor universal procedure pointer.

```
OSErr InvokeOSLGetErrDescUPP (
   AEDesc **appDescPtr,
   OSLGetErrDescUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLGetErrDescProcPtr` (page 541) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

InvokeOSLGetMarkTokenUPP

Calls an object callback get mark universal procedure pointer.

```
OSErr InvokeOSLGetMarkTokenUPP (
   const AEDesc *dContainerToken,
   DescType containerClass,
   AEDesc *result,
   OSLGetMarkTokenUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLGetMarkTokenProcPtr` (page 542) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

InvokeOSLMarkUPP

Calls an object callback mark universal procedure pointer.

```
OSErr InvokeOSLMarkUPP (
   const AEDesc *dToken,
   const AEDesc *markToken,
   long index,
   OSLMarkUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
See the `OSLMarkProcPtr` (page 544) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewAECoerceDescUPP

Creates a new universal procedure pointer to a function that coerces data stored in a descriptor.

```
AECoerceDescUPP NewAECoerceDescUPP (
   AECoerceDescProcPtr userRoutine
);
```

Return Value
See `AECoerceDescUPP` (page 552).

Discussion
See the `AECoerceDescProcPtr` (page 524) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

NewAECoercePtrUPP

Creates a new universal procedure pointer to a function that coerces data stored in a buffer.

```
AECoercePtrUPP NewAECoercePtrUPP (
   AECoercePtrProcPtr userRoutine
);
```

Return Value
See `AECoercePtrUPP` (page 552).

Discussion
See the `AECoercePtrProcPtr` (page 525) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

NewAEDisposeExternalUPP

Creates a new universal procedure pointer to a function that disposes of data stored in a buffer.

```
AEDisposeExternalUPP NewAEDisposeExternalUPP (
   AEDisposeExternalProcPtr userRoutine
);
```

Return Value
See `AEDisposeExternalUPP` (page 555).

Discussion
See the `AEDisposeExternalProcPtr` (page 527) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
AEDataModel.h

NewAEEventHandlerUPP

Creates a new universal procedure pointer to an event handler function.

```
AEEventHandlerUPP NewAEEventHandlerUPP (
   AEEventHandlerProcPtr userRoutine
);
```

Return Value
See AEEventHandlerUPP (page 555).

Discussion
See the AEEventHandlerProcPtr (page 528) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

NewAEFilterUPP

Creates a new universal procedure pointer to an Apple event filter function.

```
AEFilterUPP NewAEFilterUPP (
   AEFilterProcPtr userRoutine
);
```

Return Value
See AEFilterUPP (page 556).

Discussion
See the AEFilterProcPtr (page 530) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEInteraction.h

NewAEIdleUPP

Creates a new universal procedure pointer to an Apple event idle function.

```
AEIdleUPP NewAEIdleUPP (
   AEIdleProcPtr userRoutine
);
```

Return Value
See `AEIdleUPP` (page 556).

Discussion
See the `AEIdleProcPtr` (page 531) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

NewOSLAccessorUPP

Creates a new universal procedure pointer to an object accessor function.

```
OSLAccessorUPP NewOSLAccessorUPP (
   OSLAccessorProcPtr userRoutine
);
```

Return Value
See `OSLAccessorUPP` (page 560).

Discussion
See the `OSLAccessorProcPtr` (page 533) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLAdjustMarksUPP

Creates a new universal procedure pointer to an object callback adjust marks function.

```
OSLAdjustMarksUPP NewOSLAdjustMarksUPP (
   OSLAdjustMarksProcPtr userRoutine
);
```

Return Value
See `OSLAdjustMarksUPP` (page 561).

Discussion
See the `OSLAdjustMarksProcPtr` (page 535) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLCompareUPP

Creates a new universal procedure pointer to an object callback comparison function.

```
OSLCompareUPP NewOSLCompareUPP (
   OSLCompareProcPtr userRoutine
);
```

Return Value
See `OSLCompareUPP` (page 561).

Discussion
See the `OSLCompareProcPtr` (page 536) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLCountUPP

Creates a new universal procedure pointer to an object callback count function.

```
OSLCountUPP NewOSLCountUPP (
   OSLCountProcPtr userRoutine
);
```

Return Value
See `OSLCountUPP` (page 561).

Discussion
See the `OSLCountProcPtr` (page 538) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLDisposeTokenUPP

Creates a new universal procedure pointer to an object callback dispose token function.

```
OSLDisposeTokenUPP NewOSLDisposeTokenUPP (
   OSLDisposeTokenProcPtr userRoutine
);
```

Return Value
See `OSLDisposeTokenUPP` (page 561).

Discussion
See the `OSLDisposeTokenProcPtr` (page 539) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewOSLGetErrDescUPP

Creates a new universal procedure pointer to an object callback get error descriptor function.

```
OSLGetErrDescUPP NewOSLGetErrDescUPP (
   OSLGetErrDescProcPtr userRoutine
);
```

Return Value
See OSLGetErrDescUPP (page 562).

Discussion
See the OSLGetErrDescProcPtr (page 541) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewOSLGetMarkTokenUPP

Creates a new universal procedure pointer to an object callback get mark function.

```
OSLGetMarkTokenUPP NewOSLGetMarkTokenUPP (
   OSLGetMarkTokenProcPtr userRoutine
);
```

Return Value
See OSLGetMarkTokenUPP (page 562).

Discussion
See the OSLGetMarkTokenProcPtr (page 542) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewOSLMarkUPP

Creates a new universal procedure pointer to an object callback mark function.

```
OSLMarkUPP NewOSLMarkUPP (
   OSLMarkProcPtr userRoutine
);
```

Return Value
See `OSLMarkUPP` (page 562).

Discussion
See the `OSLMarkProcPtr` (page 544) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

vAEBuildAppleEvent

Allows you to encapsulate calls to `AEBuildAppleEvent` in a wrapper routine.

```
OSStatus vAEBuildAppleEvent (
   AEEventClass theClass,
   AEEventID theID,
   DescType addressType,
   const void *addressData,
   Size addressLength,
   SInt16 returnID,
   SInt32 transactionID,
   AppleEvent *resultEvt,
   AEBuildError *error,
   const char *paramsFmt,
   va_list args
);
```

Parameters

`theClass`
The event class for the resulting Apple event. See `AEEventClass` (page 555).

`theID`
The event id for the resulting Apple event. See `AEEventID` (page 556).

`addressType`
The address type for the addressing information described in the next two parameters: usually one of `typeApplSignature`, `typeProcessSerialNumber`, or `typeKernelProcessID`. See `DescType` (page 560).

`addressData`
A pointer to the address information.

`addressLength`
The number of bytes pointed to by the `addressData` parameter.

`returnID`
The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID.

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events.

result

A pointer to a descriptor where the resulting descriptor should be stored. See `AppleEvent` (page 559) for a description of the data type.

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See "AEBuild Error Codes" (page 563) for the syntax error codes that can be returned in this structure.

paramsFmt

An AEBuild format string describing the AppleEvent record to be created. The format of these strings is described in Technical Note TN2106, AEBuild*, AEPrint*, and Friends.

args

A variable array of arguments to be substituted into the `paramsFmt` format string. See the ANSI C Interfaces documentation for a description of the `va_list` data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Passing an argument list to `vAEBuildAppleEvent` corresponds to passing a series of individual parameters to the `AEBuildAppleEvent` (page 408) function.

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `vAEBuildAppleEvent` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

vAEBuildDesc

Allows you to encapsulate calls to `AEBuildDesc` in your own wrapper routines.

```
OSStatus vAEBuildDesc (
   AEDesc *dst,
   AEBuildError *error,
   const char *src,
   va_list args
);
```

Parameters

dst

A pointer to a descriptor where the resulting descriptor should be stored. See `AEDesc` (page 546).

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 546).

src

An *`AEBuild`* format string describing the descriptor to be created.

args

A reference to a previously defined, variable argument parameter list to use with the descriptor-string. The file `<stdarg.h>` defines macros for declaring and using the `va_list` data type.

Return Value

A numeric result code indicating the success of the call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. You can use the `error` parameter to discover information about other errors. See "Apple Event Manager Result Codes" (page 636).

Discussion

Passing an argument list to `vAEBuildDesc` corresponds to passing a series of individual parameters to the `AEBuildDesc` (page 410) function.

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple Event Manager routines to build up the descriptor piece by piece. The `vAEBuildDesc` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEHelpers.h`

vAEBuildParameters

Allows you to encapsulate calls to `AEBuildParameters` in your own `stdarg`-style wrapper routines, using techniques similar to those allowed by vsprintf.

```
OSStatus vAEBuildParameters (
   AppleEvent *event,
   AEBuildError *error,
   const char *format,
   va_list args
);
```

Parameters

event

The Apple event to which you are adding parameters. See `AppleEvent` (page 559).

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 546).

format

An *`AEBuild`* format string describing the `AEDesc` parameters to be created.

args

A reference to a previously defined, variable argument parameter list to use with the descriptor-string. The file `<stdarg.h>` defines macros for declaring and using the `va_list` data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636).

Discussion
Passing an argument list to `vAEBuildParameters` corresponds to passing a series of individual parameters to the `AEBuildParameters` (page 411) function.

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `vAEBuildParameters` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

Callbacks by Task

Callbacks When Resolving Remote Processes

`AERemoteProcessResolverCallback` (page 532)

Defines a pointer to a function the Apple Event Manager calls when the asynchronous execution of a remote process resolver completes, either due to success or failure, after a call to the `AERemoteProcessResolverScheduleWithRunLoop` function. Your callback function can use the reference passed to it to get the remote process information.

Callbacks When Creating Apple Events

`AEDisposeExternalProcPtr` (page 527)
Defines a pointer to a function the Apple Event Manager calls to dispose of a descriptor created by the `AECreateDescFromExternalPtr` function. Your callback function disposes of the buffer you originally passed to that function.

Callbacks When Sending Apple Events

`AEFilterProcPtr` (page 530)
Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your filter function determines which high-level events your application is willing to handle.

`AEIdleProcPtr` (page 531)
Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your idle function must handle update, null, operating-system, and activate events.

Coercing Apple Event Data Callbacks

`AECoerceDescProcPtr` (page 524)
Defines a pointer to a function that coerces data stored in a descriptor. Your descriptor coercion callback function coerces the data from the passed descriptor to the specified type, returning the coerced data in a second descriptor.

`AECoercePtrProcPtr` (page 525)
Defines a pointer to a function that coerces data stored in a buffer. Your pointer coercion callback routine coerces the data from the passed buffer to the specified type, returning the coerced data in a descriptor.

Handling Apple Events Callbacks

`AEEventHandlerProcPtr` (page 528)
Defines a pointer to a function that handles one or more Apple events. Your Apple event handler function performs any action requested by the Apple event, adds parameters to the reply Apple event if appropriate (possibly including error information), and returns a result code.

Object Accessor Callbacks

`OSLAccessorProcPtr` (page 533)
Your object accessor function either finds elements or properties of an Apple event object.

Object Callback Functions

`OSLAdjustMarksProcPtr` (page 535)
: Defines a pointer to an adjust marks callback function. Your adjust marks function unmarks objects previously marked by a call to your marking function.

`OSLCompareProcPtr` (page 536)
: Defines a pointer to an object comparison callback function. Your object comparison function compares one Apple event object to another or to the data for a descriptor.

`OSLCountProcPtr` (page 538)
: Defines a pointer to an object counting callback function. Your object counting function counts the number of Apple event objects of a specified class in a specified container object.

`OSLDisposeTokenProcPtr` (page 539)
: Defines a pointer to a dispose token callback function. Your dispose token function, required only if you use a complex token format, disposes of the specified token.

`OSLGetErrDescProcPtr` (page 541)
: Defines a pointer to an error descriptor callback function. Your error descriptor callback function supplies a pointer to an address where the Apple Event Manager can store the current descriptor if an error occurs during a call to the `AEResolve` function.

`OSLGetMarkTokenProcPtr` (page 542)
: Defines a pointer to a mark token callback function. Your mark token function returns a mark token.

`OSLMarkProcPtr` (page 544)
: Defines a pointer to an object marking callback function. Your object-marking function marks a specific Apple event object.

Callbacks

AECoerceDescProcPtr

Defines a pointer to a function that coerces data stored in a descriptor. Your descriptor coercion callback function coerces the data from the passed descriptor to the specified type, returning the coerced data in a second descriptor.

```
typedef OSErr (*AECoerceDescProcPtr)
(
    const AEDesc * fromDesc,
    DescType toType,
    long handlerRefcon,
    AEDesc * toDesc
);
```

If you name your function `MyAECoerceDescCallback`, you would declare it like this:

```
OSErr MyAECoerceDescCallback (
    const AEDesc * fromDesc,
    DescType toType,
    long handlerRefcon,
    AEDesc * toDesc
);
```

Parameters

fromDesc

A pointer to the descriptor that contains the data to coerce. See AEDesc (page 546).

toType

The desired descriptor type for the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See DescType (page 560).

handlerRefcon

A reference constant that is stored in the coercion dispatch table entry for the handler. The Apple Event Manager passes this value to the handler each time it calls it. The reference constant may have a value of 0.

toDesc

A pointer to a descriptor where your coercion routine must store the descriptor that contains the coerced data. See AEDesc (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). Your handler should return noErr if it successfully handled the coercion, errAECoercionFailed if it can't handle the coercion and it wants the Apple Event Manager to continue dispatching to other coercion handlers, or a nonzero result code otherwise.

Discussion

Your coercion handler should coerce the data to the desired descriptor type and return the resulting data in the descriptor specified by the result parameter.

To provide a pointer to your descriptor coercion callback function, you create a universal procedure pointer (UPP) of type AECoerceDescUPP (page 552), using the function NewAECoerceDescUPP (page 514). You can do so with code like the following:

```
AECoerceDescUPP MyCoerceDescUPP;
MyCoerceDescUPP = NewAECoerceDescUPP (&MyCoerceDescCallback)
```

You can then pass the UPP MyCoerceDescUPP as a parameter to any function that installs or removes a coercion handler, such as AEInstallCoercionHandler (page 448). If your application installs the same coercion handler to coerce more than one type of data, you can use the same UPP to install the handler multiple times.

If you wish to call your descriptor coercion callback function directly, you can use the InvokeAECoerceDescUPP (page 507) function.

After you are finished with a descriptor coercion callback function, and have removed it with the AERemoveCoercionHandler (page 468) function, you can dispose of the UPP with the DisposeAECoerceDescUPP (page 503) function. However, don't dispose of the UPP if any remaining coercion handler uses it or if you plan to install the coercion handler again.

Availability

Available in Mac OS X v10.0 and later.

Declared In

AEDataModel.h

AECoercePtrProcPtr

Defines a pointer to a function that coerces data stored in a buffer. Your pointer coercion callback routine coerces the data from the passed buffer to the specified type, returning the coerced data in a descriptor.

```
typedef OSErr (*AECoercePtrProcPtr) (
    DescType typeCode,
    const void * dataPtr,
    Size dataSize,
    DescType toType,
    long handlerRefcon,
    AEDesc * result
);
```

If you name your function `MyAECoercePtrCallback`, you would declare it like this:

```
OSErr MyAECoercePtrCallback (
    DescType typeCode,
    const void * dataPtr,
    Size dataSize,
    DescType toType,
    long handlerRefcon,
    AEDesc * result
);
```

Parameters

typeCode

The descriptor type of the original data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

dataPtr

A pointer to the data to coerce.

dataSize

The length, in bytes, of the data to coerce.

toType

The desired descriptor type for the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 581). See `DescType` (page 560).

handlerRefcon

A reference constant that is stored in the coercion dispatch table entry for the handler. The Apple Event Manager passes this value to the handler each time it calls it. The reference constant may have a value of `NULL`.

result

A pointer to a descriptor where your coercion routine must store the descriptor that contains the coerced data. If your routine cannot coerce the data, return a null descriptor. See `AEDesc` (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). Your handler should return `noErr` if it successfully handled the coercion, `errAECoercionFailed` if it can't handle the coercion and it wants the Apple Event Manager to continue dispatching to other coercion handlers, or a nonzero result code otherwise.

Discussion

To provide a pointer to your coercion callback function, you create a universal procedure pointer (UPP) of type `AECoercePtrUPP` (page 552), using the function `NewAECoercePtrUPP` (page 514). You can do so with code like the following:

```
AECoercePtrUPP MyCoercePtrUPP;
MyCoercePtrUPP = NewAECoercePtrUPP (&MyCoercePtrCallback)
```

You can then pass the UPP `MyCoercePtrUPP` as a parameter to any function that installs or removes a coercion handler, such as `AEInstallCoercionHandler` (page 448). If your application installs the same coercion handler to coerce more than one type of data, you can use the same UPP to install the handler multiple times.

If you wish to call your coercion callback function directly, you can use the `InvokeAECoercePtrUPP` (page 508) function.

After you are finished with a coercion callback function, and have removed it with the `AERemoveCoercionHandler` (page 468) function, you can dispose of the UPP with the `DisposeAECoercePtrUPP` (page 503) function. However, don't dispose of the UPP if any remaining coercion handler uses it or if you plan to install the coercion handler again.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDisposeExternalProcPtr

Defines a pointer to a function the Apple Event Manager calls to dispose of a descriptor created by the `AECreateDescFromExternalPtr` function. Your callback function disposes of the buffer you originally passed to that function.

```
typedef (void, AEDisposeExternalProcPtr)(
                const void *dataPtr,
                Size dataLength,
                long refcon);
```

If you name your function `MyAEDisposeExternalCallback`, you would declare it like this:

```
void MyAEDisposeExternalCallback (
            const void *dataPtr,
            Size dataLength,
            long refcon);
```

Parameters

dataPtr

A pointer to the data to be disposed of. The data must be immutable and must not be freed until this callback function is called.

dataLength

The length, in bytes, of the data in the *dataPtr* parameter.

refcon

A reference constant, supplied by your application in its original call to `AECreateDescFromExternalPtr` (page 418). The Apple Event Manager passes this value to your dispose function each time it calls it. The reference constant may have a value of 0.

Return Value
Your callback routine should not return a value.

Discussion
Your application must provide a universal procedure pointer to a dispose function as a parameter to the `AECreateDescFromExternalPtr` (page 418) function.

To provide a pointer to your dispose callback function, you create a universal procedure pointer (UPP) of type `AEDisposeExternalProcPtr`, using the function `NewAEDisposeExternalUPP` (page 514). You can do so with code like the following:

```
AEDisposeExternalProcPtr MyDisposeCallbackUPP;
MyDisposeCallbackUPP = NewAEDisposeExternalUPP (&MyAEDisposeExternalCallback);
```

You can then pass the UPP `MyDisposeCallbackUPP` as a parameter to the `AECreateDescFromExternalPtr` function.

If you wish to call your dispose callback function directly, you can use the `InvokeAEDisposeExternalUPP` (page 508) function.

After you are finished with your dispose callback function, you can dispose of the UPP with the `DisposeAEDisposeExternalUPP` (page 503) function. However, if you will use the same dispose function in subsequent calls to `AECreateDescFromExternalPtr`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

AEEventHandlerProcPtr

Defines a pointer to a function that handles one or more Apple events. Your Apple event handler function performs any action requested by the Apple event, adds parameters to the reply Apple event if appropriate (possibly including error information), and returns a result code.

```
typedef OSErr (*AEEventHandlerProcPtr)
(
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    long handlerRefcon
);
```

If you name your function `MyAEEventHandlerCallback`, you would declare it like this:

```
OSErr MyAEEventHandlerCallback (
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    long handlerRefcon
);
```

Parameters

`theAppleEvent`
A pointer to the Apple event to handle. See `AppleEvent` (page 559).

reply

A pointer to the default reply Apple event provided by the Apple Event Manager. See `AppleEvent` (page 559). If no reply is expected, `reply` has descriptor type `typeNull`.

handlerRefcon

The reference constant stored in the Apple event dispatch table when you install the handler function for the Apple event. You can store any 32-bit value in the dispatch table and use it any way you want when the handler is called. The reference constant may have a value of `NULL`.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). Your handler should always return `noErr` if it successfully handled the Apple event. If an error occurs, your handler should return either `errAEEventNotHandled` or some other nonzero result code. For more information, see the Discussion section.

Discussion

An Apple event handler should extract any parameters and attributes from the Apple event, perform the requested action, and add parameters to the reply Apple event if appropriate. You must provide an Apple event handler for each Apple event your application supports. The `AEProcessAppleEvent` (page 457) function calls one of your Apple event handlers when it processes an Apple event.

If an error occurs because your application cannot understand the event, return `errAEEventNotHandled`, so that the Apple Event Manager may be able to find another handler to handle the event. If the error occurs because the event is impossible to handle as specified, return the result code returned by whatever function caused the failure, or whatever other result code is appropriate.

For example, suppose your application receives a `kAEGetData` event that requests the name of the current printer, and your application cannot handle such an event. In this situation, you should return `errAEEventNotHandled` so that another handler available to the Apple Event Manager can have a chance to handle the event. This strategy allows users to take advantage of system capabilities from within your application via system handlers.

If your Apple event handler calls the `AEResolve` (page 473) function and `AEResolve` calls an object accessor function in the system object accessor dispatch table, your Apple event handler may not recognize the descriptor type of the token returned by the function. In this case, your handler should return the result code `errAEUnknownObjectType`. When your handler returns this result code, the Apple Event Manager attempts to locate a system Apple event handler that can recognize the token.

For additional information on dealing with error conditions, see `OSLGetErrDescProcPtr` (page 541).

To provide a pointer to your event handler callback function, you create a universal procedure pointer (UPP) of type `AEEventHandlerUPP` (page 555), using the function `NewAEEventHandlerUPP` (page 515). You can do so with code like the following:

```
AEEventHandlerUPP MyEventHandlerUPP;
MyEventHandlerUPP = NewAEEventHandlerUPP (&MyEventHandlerCallback)
```

You can then pass the UPP `MyEventHandlerUPP` as a parameter to any function that installs or removes a handler, such as `AEInstallEventHandler` (page 449). If your application installs the same event handler to handle more than one kind of event (more than one pair of event class and event ID), you can use the same UPP to install the handler multiple times.

If you wish to call your event handler callback function directly, you can use the `InvokeAEEventHandlerUPP` (page 509) function.

After you are finished with an event handler callback function, and have removed it with the `AERemoveEventHandler` (page 469) function, you can dispose of the UPP with the `DisposeAEEventHandlerUPP` (page 504) function. However, don't dispose of the UPP if any remaining handler uses it or if you plan to install the handler again.

Version Notes
Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive an Apple event—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEFilterProcPtr

Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your filter function determines which high-level events your application is willing to handle.

```
typedef Boolean (*AEFilterProcPtr) (
    EventRecord * theEvent,
    long returnID,
    long transactionID,
    const AEAddressDesc * sender
);
```

If you name your function `MyAEFilterCallback`, you would declare it like this:

```
Boolean MyAEFilterCallback (
    EventRecord * theEvent,
    long returnID,
    long transactionID,
    const AEAddressDesc * sender
);
```

Parameters

`theEvent`
A pointer to the event record for a high-level event. The next three parameters contain valid information only if the event is an Apple event. See the Event Manager documentation for a description of the `EventRecord` data type.

`returnID`
Return ID for the Apple event.

`transactionID`
Transaction ID for the Apple event.

`sender`
A pointer to the address of the process that sent the Apple event. See `AEAddressDesc` (page 551).

Return Value
Your filter routine returns `TRUE` to accept the Apple event or `FALSE` to filter it out.

Discussion
If your application provides a universal procedure pointer to a reply filter function as a parameter to the `AESend` (page 476) function, the reply filter function can indicate any high-level events that it is willing to handle while your application is waiting for a reply.

If your filter function returns `true`, the Apple Event Manager will dispatch the event through the standard dispatch mechanism (equivalent to calling `AEProcessAppleEvent` (page 457)).

To provide a pointer to your reply filter callback function, you create a universal procedure pointer (UPP) of type `AEFilterUPP` (page 556), using the function `NewAEFilterUPP` (page 515). You can do so with code like the following:

```
AEFilterUPP MyReplyFilterUPP;
MyReplyFilterUPP = NewAEFilterUPP (&MyReplyFilterCallback)
```

You can then pass the UPP `MyReplyFilterUPP` as a parameter to the `AESend` function.

If you wish to call your filter callback function directly, you can use the `InvokeAEFilterUPP` (page 509) function.

After you are finished with your filter callback function, you can dispose of the UPP with the `DisposeAEFilterUPP` (page 504) function. However, if you will use the same filter function in subsequent calls to `AESend`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEIdleProcPtr

Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your idle function must handle update, null, operating-system, and activate events.

```
typedef Boolean (*AEIdleProcPtr) (
    EventRecord * theEvent,
    long * sleepTime,
    RgnHandle * mouseRgn
);
```

If you name your function `MyAEIdleCallback`, you would declare it like this:

```
Boolean MyAEIdleCallback (
    EventRecord * theEvent,
    long * sleepTime,
    RgnHandle * mouseRgn
);
```

Parameters

`theEvent`

A pointer to the event record of the event to process. See the Event Manager documentation for a description of the `EventRecord` data type.

sleepTime

A pointer to a value that specifies the amount of time (in ticks) your application is willing to relinquish the processor if no events are pending.

mouseRgn

A pointer to a value that specifies a screen region that determines the conditions under which your application is to receive notice of mouse-moved events. See the QuickDraw Manager documentation for a description of the `RgnHandle` data type.

Return Value
Your idle routine returns `TRUE` if your application is no longer willing to wait for a reply from the server or for the user to bring the application to the front. It returns `FALSE` if your application is still willing to wait.

Discussion
If your application provides a pointer to an idle function as a parameter to the `AESend` (page 476) function or the `AEInteractWithUser` (page 453) function, the Apple Event Manager will call the idle function to handle any update event, null event, operating-system event, or activate event received for your application while it is waiting for a reply.

To provide a pointer to your idle callback function, you create a universal procedure pointer (UPP) of type `AEIdleUPP` (page 556), using the function `NewAEIdleUPP` (page 515). You can do so with code like the following:

```
AEIdleUPP MyIdleUPP;
MyIdleUPP = NewAEIdleUPP (&MyIdleCallback)
```

You can then pass the UPP `MyIdleUPP` as a parameter to either the `AESend` function or the `AEInteractWithUser` function.

If you wish to call your idle callback function directly, you can use the `InvokeAEIdleUPP` (page 509) function.

After you are finished with your idle callback function, you can dispose of the UPP with the `DisposeAEIdleUPP` (page 504) function. However, if you will use the same idle function in subsequent calls to `AESend` or `AEInteractWithUser`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AERemoteProcessResolverCallback

Defines a pointer to a function the Apple Event Manager calls when the asynchronous execution of a remote process resolver completes, either due to success or failure, after a call to the `AERemoteProcessResolverScheduleWithRunLoop` function. Your callback function can use the reference passed to it to get the remote process information.

```
typedef (void, AERemoteProcessResolverCallback)(
                AERemoteProcessResolverRef ref,
                void *info);
```

If you name your function `MyAERemoteProcessCallback`, you would declare it like this:

```
void MyAERemoteProcessCallback (
```

```
            AERemoteProcessResolverRef ref,
            void *info);
```

Parameters

ref

A reference of type `AERemoteProcessResolverRef` (page 557) you can query to obtain the remote process information. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 420).

info

An untyped pointer your application can use to pass information it needs when resolving remote processes. The application originally supplies this pointer in the `AERemoteProcessResolverContext` (page 547) structure in the `ctx` parameter) when it calls the `AERemoteProcessResolverScheduleWithRunLoop` function.

Return Value
Your callback routine should not return a value.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

OSLAccessorProcPtr

Your object accessor function either finds elements or properties of an Apple event object.

```
typedef OSErr (*OSLAccessorProcPtr) (
    DescType desiredClass,
    const AEDesc * container,
    DescType containerClass,
    DescType form,
    const AEDesc * selectionData,
    AEDesc * value,
    long accessorRefcon
);
```

If you name your function `MyObjectAccessorCallback`, you would declare it like this:

```
OSErr MyObjectAccessorCallback (
    DescType desiredClass,
    const AEDesc * container,
    DescType containerClass,
    DescType form,
    const AEDesc * selectionData,
    AEDesc * value,
    long accessorRefcon
);
```

Parameters

desiredClass

The object class of the desired Apple event object or objects. Constants for object class IDs are described in "Object Class ID Constants" (page 599). See `DescType` (page 560).

container
A pointer to a descriptor that specifies the container of the desired Apple event object or objects. See `AEDesc` (page 546).

containerClass
The object class of the container. Constants for object class IDs are described in "Object Class ID Constants" (page 599). See `DescType` (page 560).

form
The key form specified by the object specifier being resolved. Constants for key form are described in "Key Form and Descriptor Type Object Specifier Constants" (page 590). See `DescType` (page 560).

selectionData
A pointer to a descriptor containing the key data specified by the object specifier being resolved. See `AEDesc` (page 546).

value
A pointer to a descriptor where your object accessor routine stores a descriptor that identifies the found object. See `AEDesc` (page 546).

accessorRefcon
A reference constant. The Apple Event Manager passes this value to your object accessor function each time it calls it. The reference constant may have a value of 0.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). Your object accessor function should return `noErr` if it successfully located the requested object and `errAEEventNotHandled` if it could not locate the object. When the Apple Event Manager receives the result code `errAEEventNotHandled` after calling an object accessor function, it attempts to use other methods of locating the requested objects, such as calling an equivalent system object accessor function.

Discussion
To resolve an object specifier, your application calls the `AEResolve` (page 473) function. `AEResolve` in turn calls application-defined object accessor functions to locate specific Apple event objects and properties in the application's data structures. Your application provides one or more object accessor functions that can locate all the element classes and properties it supports.

Each object accessor function provided by your application should either find elements or properties of an Apple event object. The `AEResolve` function uses the object class ID of the specified Apple event object and the descriptor type of the token that identifies the object's container to determine which object accessor function to call. To install an object accessor function, use the `AEInstallObjectAccessor` (page 451) function.

To provide a pointer to your object accessor callback function, you create a universal procedure pointer (UPP) of type `OSLAccessorUPP` (page 560), using the function `NewOSLAccessorUPP` (page 516). You can do so with code like the following:

```
AEObjectAccessorUPP MyObjectAccessorUPP;
MyObjectAccessorUPP = NewAEObjectAccessorUPP (&MyObjectAccessorCallback)
```

You can then pass the UPP `MyObjectAccessorUPP` as a parameter to any function that installs or removes an object accessor, such as `AEInstallObjectAccessor` (page 451). If your application installs the same object accessor to handle more than one kind of object class or property of an Apple event, you can use the same UPP to install the accessor multiple times.

If you wish to call your object accessor callback function directly, you can use the `InvokeOSLAccessorUPP` (page 510) function.

After you are finished with an object accessor callback function, and have removed it with the `AERemoveObjectAccessor` (page 470) function, you can dispose of the UPP with the `DisposeOSLAccessorUPP` (page 504) function. However, don't dispose of the UPP if any remaining accessor function uses it or if you plan to install the accessor function again.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLAdjustMarksProcPtr

Defines a pointer to an adjust marks callback function. Your adjust marks function unmarks objects previously marked by a call to your marking function.

```
typedef OSErr (*OSLAdjustMarksProcPtr)
(
    long newStart,
    long newStop,
    const AEDesc * markToken
);
```

If you name your function `MyAdjustMarksCallback`, you would declare it like this:

```
OSErr MyAdjustMarksCallback (
    long newStart,
    long newStop,
    const AEDesc * markToken
);
```

Parameters

newStart
The mark count value (provided when the `MyAdjustMarksCallback` callback function was called to mark the object) for the first object in the new set of marked objects.

newStop
The mark count value (provided when the `MyAdjustMarksCallback` callback function was called to mark the object) for the last object in the new set of marked objects.

markToken
A pointer to the mark token for the marked objects. (Token is defined in `AEDisposeToken` (page 425). See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). Your adjust marks function should return `noErr` if it successfully adjusted the marks and `errAEEventNotHandled` if it could not locate the object. When the Apple Event Manager gets an error result of `errAEEventNotHandled`, it attempts to adjust the marks by calling the equivalent system mark-adjusting function.

Discussion
When the Apple Event Manager needs to identify either a range of elements or the absolute position of an element in a group of Apple event objects that pass a test, it can use your application's mark-adjusting function to unmark objects previously marked by a call to your marking function.

For example, suppose an object specifier specifies `any row in the table "MyCustomers" for which the City column is "San Francisco"`. The Apple Event Manager first uses the appropriate object accessor function to locate all the rows in the table for which the City column is `"San Francisco"` and calls the application's marking function repeatedly to mark them. It then generates a random number between 1 and the number of rows it found that passed the test and calls the application's mark-adjusting function to unmark all the rows whose mark count does not match the randomly generated number. If the randomly chosen row has a mark count value of 5, the Apple Event Manager passes the value 5 to the mark-adjusting function in both the `newStart` parameter and the `newStop` parameter, and passes the current mark token in the `markToken` parameter.

When the Apple Event Manager calls your `MyAdjustMarksCallback` function, your application must dispose of any data structures that it created to mark the previously marked objects.

To provide a pointer to your adjust marks callback function, you create a universal procedure pointer (UPP) of type `OSLAdjustMarksUPP` (page 561), using the function `NewOSLAdjustMarksUPP` (page 516). You can do so with code like the following:

```
OSLAdjustMarksUPP MyAdjustMarksUPP;
MyAdjustMarksUPP = NewOSLAdjustMarksUPP (&MyAdjustMarksCallback)
```

You can then pass the UPP `MyAdjustMarksUPP` as a parameter to the `AESetObjectCallbacks` (page 480) function or the `AEInstallSpecialHandler` (page 452) function.

If you wish to call your adjust marks callback function directly, you can use the `InvokeOSLAdjustMarksUPP` (page 510) function.

After you are finished with your adjust marks callback function, you can dispose of the UPP with the `DisposeOSLAdjustMarksUPP` (page 505) function. However, if you will use the same adjust marks function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCompareProcPtr

Defines a pointer to an object comparison callback function. Your object comparison function compares one Apple event object to another or to the data for a descriptor.

```
typedef OSErr (*OSLCompareProcPtr) (
    DescType oper,
    const AEDesc * obj1,
    const AEDesc * obj2,
    Boolean * result
);
```

If you name your function `MyCompareObjectsCallback`, you would declare it like this:

```
OSErr MyCompareObjectsCallback (
    DescType oper,
    const AEDesc * obj1,
    const AEDesc * obj2,
    Boolean * result
);
```

Parameters

oper

A comparison operator that specifies the type of comparison to perform. The available comparison operators are described in "Comparison Operator Constants" (page 574). For related information, see the function `CreateCompDescriptor` (page 498). See `DescType` (page 560).

obj1

A pointer to a token describing the first Apple event object to compare. (Token is defined in `AEDisposeToken` (page 425). See `AEDesc` (page 546).

obj2

A pointer to a token or some other descriptor that specifies either an Apple event object or a value to compare to the Apple event object specified by the `obj1` parameter. See `AEDesc` (page 546).

result

A pointer to a Boolean value where your object comparison function stores a value indicating the result of the comparison operation. You store `TRUE` if the values of the `obj1` and `obj2` parameters have the relationship specified by the `comparisonOperator` parameter; otherwise, you store `FALSE`.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). Your object comparison function should return `noErr` if it successfully compared the objects and `errAEEventNotHandled` if it can't compare the objects. When the Apple Event Manager gets an error result of `errAEEventNotHandled`, it attempts to use other methods of comparing the specified objects, such as calling an equivalent system object comparison function.

Discussion

The Apple Event Manager calls your object comparison function when, in the course of resolving an object specifier, the manager needs to compare an Apple event object with another object or with a value in a descriptor.

If you want the Apple Event Manager to help your application resolve object specifiers of key form `formTest` (and if your application doesn't specify `kAEIDoWhose` as described in "Callback Constants for the AEResolve Function" (page 571)), you should provide an object-counting function, as described in `OSLCountProcPtr` (page 538), and an object comparison function.

It is up to your application to interpret the comparison operators it receives. The meaning of comparison operators differs according to the Apple event objects being compared, and not all comparison operators apply to all object classes. The available comparison operators are described in "Comparison Operator Constants" (page 574).

To provide a pointer to your object comparison callback function, you create a universal procedure pointer (UPP) of type `OSLCompareUPP` (page 561), using the function `NewOSLCompareUPP` (page 517). You can do so with code like the following:

```
OSLCompareObjectsUPP MyCompareObjectsUPP;
MyCompareObjectsUPP = NewOSLCompareObjectsUPP(&MyCompareObjectsCallback)
```

You can then pass the UPP `MyCompareObjectsUPP` as a parameter to the `AESetObjectCallbacks` (page 480) function or the `AEInstallSpecialHandler` (page 452) function.

If you wish to call your object comparison callback function directly, you can use the `InvokeOSLCompareUPP` (page 511) function.

After you are finished with your object comparison callback function, you can dispose of the UPP with the `DisposeOSLCompareUPP` (page 505) function. However, if you will use the same object comparison function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCountProcPtr

Defines a pointer to an object counting callback function. Your object counting function counts the number of Apple event objects of a specified class in a specified container object.

```
typedef OSErr (*OSLCountProcPtr) (
    DescType desiredType,
    DescType containerClass,
    const AEDesc * container,
    long * result
);
```

If you name your function `MyCountObjectsCallback`, you would declare it like this:

```
OSErr MyCountObjectsCallback (
    DescType desiredType,
    DescType containerClass,
    const AEDesc * container,
    long * result
);
```

Parameters

desiredType
The object class of the Apple event objects to be counted. See `DescType` (page 560).

containerClass
The object class of the container for the Apple event objects to be counted. See `DescType` (page 560).

container
A pointer to a token that identifies the container for the Apple event objects to be counted. (Token is defined in `AEDisposeToken` (page 425). See `AEDesc` (page 546).

result

A pointer to a variable where your object-counting function stores the number of Apple objects of the specified class in the specified container.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). Your object counting function should return `noErr` if it successfully counted the objects and `errAEEventNotHandled` if it can't count the objects. When the Apple Event Manager receives the result code `errAEEventNotHandled` after calling an object counting function, it attempts to use other methods of counting the specified objects, such as calling an equivalent system object counting function.

Discussion
If you want the Apple Event Manager to help your application resolve object specifiers of key form `formTest` (and if your application doesn't specify `kAEIDoWhose` as described in "Callback Constants for the AEResolve Function" (page 571)), you should provide an object comparison function, as described in `OSLCompareProcPtr` (page 536), and an object-counting function.

The Apple Event Manager calls your object-counting function when, in the course of resolving an object specifier, the manager requires a count of the number of Apple event objects of a given class in a given container.

To provide a pointer to your object counting callback function, you create a universal procedure pointer (UPP) of type `OSLCountUPP` (page 561), using the function `NewOSLCountUPP` (page 517). You can do so with code like the following:

```
OSLCountObjectsUPP MyCountObjectsUPP;
MyCountObjectsUPP = NewOSLCountObjectsUPP (&MyCountObjectsCallback)
```

You can then pass the UPP `MyCountObjectsUPP` as a parameter to the `AESetObjectCallbacks` (page 480) function or the `AEInstallSpecialHandler` (page 452) function.

If you wish to call your object counting callback function directly, you can use the `InvokeOSLCountUPP` (page 511) function.

After you are finished with your object counting callback function, you can dispose of the UPP with the `DisposeOSLCountUPP` (page 505) function. However, if you will use the same object counting function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLDisposeTokenProcPtr

Defines a pointer to a dispose token callback function. Your dispose token function, required only if you use a complex token format, disposes of the specified token.

```
typedef OSErr (*OSLDisposeTokenProcPtr)
(
    AEDesc * unneededToken
);
```

If you name your function `MyDisposeTokenCallback`, you would declare it like this:

```
OSErr MyDisposeTokenCallback (
    AEDesc * unneededToken
);
```

Parameters

unneededToken

A pointer to the token to dispose of. (Token is defined in `AEDisposeToken` (page 425).) On successful return, your function must set this to the null descriptor. See `AEDesc` (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). Your token disposal function should return `noErr` if it successfully disposed of the token and `errAEEventNotHandled` if it can't dispose of the token. When the Apple Event Manager receives the result code `errAEEventNotHandled` after calling a token disposal function, it attempts to use other methods of disposing of the specified token, such as calling an equivalent system token disposal function if one is available or, if that fails, by calling `AEDisposeDesc` (page 424).

Discussion

The Apple Event Manager calls your token disposal function whenever it needs to dispose of a token. It also calls your disposal function when your application calls the `AEDisposeToken` (page 425) function. If your application does not provide a token disposal function, the Apple Event Manager calls `AEDisposeDesc` (page 424) instead.

Your token disposal function must be able to dispose of all of the token types used by your application.

If your application supports marking, a call to `MyDisposeTokenCallback` to dispose of a mark token lets your application know that it can unmark the objects marked with that mark token, as described in the Discussion section for `OSLGetMarkTokenProcPtr` (page 542).

To provide a pointer to your token disposal callback function, you create a universal procedure pointer (UPP) of type `OSLDisposeTokenUPP` (page 561), using the function `NewOSLDisposeTokenUPP` (page 517). You can do so with code like the following:

```
OSLDisposeTokenUPP MyDisposeTokenUPP;
MyDisposeTokenUPP = NewOSLDisposeTokenUPP (&MyDisposeTokenCallback)
```

You can then pass the UPP `MyDisposeTokenUPP` as a parameter to the `AESetObjectCallbacks` (page 480) function or the `AEInstallSpecialHandler` (page 452) function.

If you wish to call your token disposal callback function directly, you can use the `InvokeOSLDisposeTokenUPP` (page 512) function.

After you are finished with your token disposal callback function, you can dispose of the UPP with the `DisposeOSLDisposeTokenUPP` (page 506) function. However, if you will use the same token disposal function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetErrDescProcPtr

Defines a pointer to an error descriptor callback function. Your error descriptor callback function supplies a pointer to an address where the Apple Event Manager can store the current descriptor if an error occurs during a call to the `AEResolve` function.

```
typedef OSErr (*OSLGetErrDescProcPtr)
(
    AEDesc ** appDescPtr
);
```

If you name your function `MyGetErrorDescCallback`, you would declare it like this:

```
OSErr MyGetErrorDescCallback (
    AEDesc ** appDescPtr
);
```

Parameters

appDescPtr

A pointer to a pointer to a descriptor address. Your error descriptor callback function supplies a pointer to an address of a descriptor where the Apple Event Manager can store the current descriptor if an error occurs. See `AEDesc` (page 546).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). Your error descriptor function should return `noErr` if it completes successfully and a nonzero error value if it is unsuccessful. If it returns a nonzero value, the Apple Event Manager continues to resolve the object specifier as if it had never called the error callback function.

Discussion
Your get error descriptor callback function simply supplies a pointer to an address. Shortly after your application calls the `AEResolve` (page 473) function, the Apple Event Manager calls your get error descriptor callback function and writes a null descriptor to the address supplied by your callback, overwriting whatever was there previously.

If an error occurs during the resolution of the object specifier, the Apple Event Manager calls your get error descriptor callback function again and writes the descriptor it is currently working with—often an object specifier—to the address supplied by your callback. If `AEResolve` returns an error during the resolution of an object specifier, this address contains the descriptor responsible for the error.

You should always write a null descriptor at the address provided by your get error descriptor callback function before calling `AEResolve`. When recovering from an error, the Apple Event Manager, never writes to the address you provide unless it already contains a null descriptor. You may wish to maintain a single global variable of type `AEDesc` and have your get error descriptor callback function always provide the address of that variable.

After `AEResolve` returns, if your error descriptor is not the null descriptor, you are responsible for disposing of it.

To provide a pointer to your get error descriptor callback function, you create a universal procedure pointer (UPP) of type `OSLGetErrDescUPP` (page 562), using the function `NewOSLGetErrDescUPP` (page 518). You can do so with code like the following:

```
OSLGetErrorDescUPP MyGetErrorDescUPP;
MyGetErrorDescUPP = NewOSLGetErrorDescUPP (&MyGetErrorDescCallback)
```

You can then pass the UPP `MyGetErrorDescUPP` as a parameter to the `AESetObjectCallbacks` (page 480) function or the `AEInstallSpecialHandler` (page 452) function.

If you wish to call your get error descriptor callback function directly, you can use the `InvokeOSLGetErrDescUPP` (page 512) function.

After you are finished with your get error descriptor callback function, you can dispose of the UPP with the `DisposeOSLGetErrDescUPP` (page 506) function. However, if you will use the same get error descriptor callback function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetMarkTokenProcPtr

Defines a pointer to a mark token callback function. Your mark token function returns a mark token.

```
typedef OSErr (*OSLGetMarkTokenProcPtr)
(
    const AEDesc * dContainerToken,
    DescType containerClass,
    AEDesc * result
);
```

If you name your function `MyGetMarkTokenCallback`, you would declare it like this:

```
OSErr MyGetMarkTokenCallback (
    const AEDesc * dContainerToken,
    DescType containerClass,
    AEDesc * result
);
```

Parameters

dContainerToken

A pointer to the Apple event object that contains the elements to be marked with the mark token. (Token is defined in AEDisposeToken (page 425). See AEDesc (page 546).

containerClass

The object class of the container that contains the objects to be marked. See DescType (page 560).

result

A pointer to a descriptor where your mark token function should return a mark token. If your function can't return a mark token, it should return a null descriptor. See AEDesc (page 546).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 636). Your mark token function should return noErr if it successfully supplies a mark token and errAEEventNotHandled if it fails to supply a mark token. When the Apple Event Manager gets an error result of errAEEventNotHandled after calling a mark token function, it attempts to get a mark token by calling the equivalent system marking callback function.

Discussion

To get a mark token, the Apple Event Manager calls your mark token function. Like other tokens, the mark token returned can be a descriptor of any type; however, unlike other tokens, a mark token identifies the way your application will mark Apple event objects during the current session while resolving a single object specifier that specifies the key form formTest.

A mark token is valid until the Apple Event Manager either disposes of it by calling AEDisposeToken (page 425) or returns it as the result of the AEResolve (page 473) function. If the final result of a call to AEResolve is a mark token, the Apple event objects currently marked for that mark token are those specified by the object specifier passed to AEResolve, and your application can proceed to do whatever the Apple event has requested. Note that your application is responsible for disposing of a final mark token with a call to AEDisposeToken, just as for any other final token.

If your application supports marking, it should also provide a token disposal function modeled after the token disposal function described in OSLDisposeTokenProcPtr (page 539). When the Apple Event Manager calls AEDisposeToken to dispose of a mark token that is not the final result of a call to AEResolve, the subsequent call to your token disposal function lets you know that you can unmark the Apple event objects marked with that mark token. A call to AEDisposeDesc to dispose of a mark token (which would occur if you did not provide a token disposal function) would go unnoticed.

To provide a pointer to your mark token callback function, you create a universal procedure pointer (UPP) of type OSLGetMarkTokenUPP (page 562), using the function NewOSLGetMarkTokenUPP (page 518). You can do so with code like the following:

```
OSLGetMarkTokenUPP MyGetMarkTokenUPP;
MyGetMarkTokenUPP = NewOSLGetMarkTokenUPP (&MyGetMarkTokenCallback)
```

You can then pass the UPP MyGetMarkTokenUPP as a parameter to the AESetObjectCallbacks (page 480) function or the AEInstallSpecialHandler (page 452) function.

If you wish to call your mark token callback function directly, you can use the InvokeOSLGetMarkTokenUPP (page 513) function.

After you are finished with your mark token callback function, you can dispose of the UPP with the DisposeOSLGetMarkTokenUPP (page 506) function. However, if you will use the same mark token function in subsequent calls to the function AESetObjectCallbacks or the function AEInstallSpecialHandler, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

OSLMarkProcPtr

Defines a pointer to an object marking callback function. Your object-marking function marks a specific Apple event object.

```
typedef OSErr (*OSLMarkProcPtr) (
    const AEDesc * dToken,
    const AEDesc * markToken,
    long index
);
```

If you name your function MyMarkCallback, you would declare it like this:

```
OSErr MyMarkCallback (
    const AEDesc * dToken,
    const AEDesc * markToken,
    long index
);
```

Parameters

dToken

A pointer to the token for the Apple event object to be marked. (Token is defined in AEDisposeToken (page 425). See AEDesc (page 546).

markToken

A pointer to the mark token used to mark the Apple event object. See AEDesc (page 546).

index

The number of times your MyMarkCallback function has been called for the current mark token (that is, the number of Apple event objects that have so far passed the test, including the element to be marked).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 636). Your object marking function should return noErr if it successfully marks the Apple event object and errAEEventNotHandled if it fails to mark the object. When the Apple Event Manager gets an error result of errAEEventNotHandled after calling an object marking function, it attempts to get mark the object by calling the equivalent system object marking function.

Discussion
To mark an Apple event object using the current mark token, the Apple Event Manager calls the object-marking function provided by your application. In addition to marking the specified object, your MyMarkCallback function should record the mark count for each object that it marks. The mark count recorded for each marked object allows your application to determine which of a set of marked tokens pass a test, as described in the Discussion section for the OSLAdjustMarksProcPtr (page 535) function.

To provide a pointer to your mark callback function, you create a universal procedure pointer (UPP) of type OSLMarkUPP (page 562), using the function NewOSLMarkUPP (page 518). You can do so with code like the following:

```
OSLMarkUPP MyMarkUPP;
MyMarkUPP = NewOSLMarkUPP (&MyMarkCallback)
```

You can then pass the UPP `MyMarkUPP` as a parameter to the `AESetObjectCallbacks` (page 480) function or the `AEInstallSpecialHandler` (page 452) function.

If you wish to call your mark callback function directly, you can use the `InvokeOSLMarkUPP` (page 513) function.

After you are finished with your mark callback function, you can dispose of the UPP with the `DisposeOSLMarkUPP` (page 507) function. However, if you will use the same mark function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

Data Types

AEArrayData

Stores array information to be put into a descriptor list with the `AEPutArray` function or extracted from a descriptor list with the `AEGetArray` function.

```
union AEArrayData {
    short kAEDataArray[1];
    char kAEPackedArray[1];
    Handle kAEHandleArray[1];
    AEDesc kAEDescArray[1];
    AEKeyDesc kAEKeyDescArray[1];
};
typedef union AEArrayData AEArrayData;
```

Discussion
When your application calls the `AEPutArray` (page 459) function to put information into a descriptor list or the `AEGetArray` (page 428) function to get information from a descriptor list, it uses an to store the information. The type of array depends on the data for the array, as specified by one of the constants described in "Data Array Constants" (page 580).

Array items in Apple event arrays of type `kAEDataArray`, `kAEPackedArray`, or `kAEHandleArray` must be factored—that is, contained in a factored descriptor list. Before adding array items to a factored descriptor list, you should provide both a pointer to the data that is common to all array items and the size of that common data when you first call `AECreateList` (page 419) to create a factored descriptor list. When you call `AEPutArray` to add the array data to such a descriptor list, the Apple Event Manager automatically isolates the common data you specified in the call to `AECreateList`.

When you call `AEGetArray` or `AEPutArray`, you specify a pointer of data type `AEArrayDataPointer` that points to a buffer containing the data for the array.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEBuildError

Defines a structure for storing additional error code information for "AEBuild" routines.

```
struct AEBuildError {
    AEBuildErrorCode fError;
    UInt32 fErrorPos;
};
typedef struct AEBuildError AEBuildError;
```

Fields

`fError`
The error code. See "AEBuild Error Codes" (page 563) for a list of errors.

`fErrorPos`
The character position where the parser detected the error.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEHelpers.h

AEDesc

Stores data and an accompanying descriptor type to form the basic building block of all Apple Events.

```
struct AEDesc {
    DescType descriptorType;
    AEDataStorage dataHandle;
};
typedef struct AEDesc AEDesc;
```

Fields

`descriptorType`
A four-character code of type `DescType` (page 560) that indicates the type of data in the structure. See `DescType` (page 560).

`dataHandle`
An opaque storage type that points to the storage for the descriptor data. Your application doesn't access this data directly—rather, it calls one of the functions `AEGetDescDataSize` (page 434), `AEGetDescData` (page 432), or `AEReplaceDescData` (page 472). See `AEDataStorage` (page 553).

Discussion
The Apple Event Manager uses one or more descriptors to construct Apple event attributes and parameters, object specifiers, tokens, and many other types of data it works with. (Token is defined in `AEDisposeToken` (page 425).) A descriptor consists of an opaque data storage container and a descriptor type that identifies the type of the data stored in the descriptor.

The descriptor type is a structure of type `DescType`, which in turn is of data type `ResType`—that is, a four-character code. "Descriptor Type Constants" (page 581) lists the constants for the basic descriptor types used by the Apple Event Manager. For information about descriptor types used with object specifiers, see "Key Form and Descriptor Type Object Specifier Constants" (page 590).

Version Notes
Prior to Carbon, the `AEDataStorage` (page 553) data type was defined as follows:

```
typedef Handle AEDataStorage;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEKeyDesc

Associates a keyword with a descriptor to form a keyword-specified descriptor.

```
struct AEKeyDesc {
    AEKeyword descKey;
    AEDesc descContent;
};
typedef struct AEKeyDesc AEKeyDesc;
```

Fields

`descKey`
A four-character code of type `AEKeyword` (page 556) that uniquely identifies the key that is associated with the data in the structure. Some keyword constants are described in "Keyword Attribute Constants" (page 593) and "Keyword Parameter Constants" (page 595). See `AEKeyword` (page 556).

`descContent`
A descriptor of type `AEDesc` (page 546) that stores the keyword descriptor data. See `AEDesc` (page 546).

Discussion
The Apple Event Manager uniquely identifies the various parts of an Apple event by means of keywords associated with corresponding descriptors. A keyword is an arbitrary constant of type `AEKeyword` (page 556) that represents a four-character code.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERemoteProcessResolverContext

Supplied as a parameter when performing asynchronous resolution of remote processes.

```
struct AERemoteProcessResolverContext {
    CFIndex version;
    void * info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct AERemoteProcessResolverContext AERemoteProcessResolverContext;
```

Fields

`version`
: This should be set to zero (0).

`info`
: A pointer to arbitrary information. The pointer is retained and passed to the callback, allowing you to provide information to that routine.

`retain`
: A prototype for a function callback that retains the specified data. Called on the info pointer. This field may be `NULL`.

`release`
: A prototype for a function callback that releases the specified data. Called on the info pointer. This field may be `NULL`.

`copyDescription`
: A prototype for a function callback that provides a description of the specified data. Called on the info pointer. This field may be `NULL`.

Discussion

When you call `AERemoteProcessResolverScheduleWithRunLoop` (page 467) for asynchronous resolution, you supply a reference to a structure of this type, along with a reference to a callback routine, defined by `AERemoteProcessResolverCallback` (page 532). The context is copied and the info pointer retained. When the callback is made, the info pointer is passed to the callback.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`AppleEvents.h`

ccntTokenRecord

Stores token information used by the AEResolve function while locating a range of objects.

```
struct ccntTokenRecord {
    DescType tokenClass;
    AEDesc token;
};
typedef struct ccntTokenRecord ccntTokenRecord;
```

Fields

`tokenClass`
: The class ID of the container represented by the `token` parameter. See `DescType` (page 560).

`token`
: A token for the current container. (Token is defined in `AEDisposeToken` (page 425). See `AEDesc` (page 546).

Discussion
When the `AEResolve` (page 473) function calls an object accessor function to locate a range of objects, the Apple Event Manager replaces the descriptor of type `typeCurrentContainer` with a token for the container of each boundary object. When using `AEResolve` to resolve the object specifier, your application doesn't need to examine the contents of this token, because the Apple Event Manager keeps track of it.

If your application attempts to resolve some or all of the object specifier without calling `AEResolve`, the application may need to examine the token before it can locate the boundary objects. The token provided by the Apple Event Manager for a boundary object's container is a descriptor of type `typeToken` whose data storage pointer refers to a structure of type `ccntTokenRecord`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

IntlText

International text consists of an ordered series of bytes, beginning with a 4-byte language code and a 4-byte script code that together determine the format of the bytes that follow. (**Deprecated.** Use Unicode text instead.)

```
struct IntlText {
    ScriptCode theScriptCode;
    LangCode theLangCode;
    char theText[1];
};
typedef struct IntlText IntlText;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AERegistry.h`

OffsetArray

Specifies offsets of ranges of text. Not typically used by developers.

```
struct OffsetArray {
    sort fNumOfOffsets;
    long fOffset[1];
};
typedef struct OffsetArray OffsetArray;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AERegistry.h`

TextRange

Specifies a range of text. Not typically used by developers.

```
struct TextRange {
    long fStart;
    long fEnd;
    short fHiliteStyle;
};
typedef struct TextRange TextRange;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AERegistry.h

TextRangeArray

Specifies an array of text ranges. Not typically used by developers.

```
struct TextRangeArray {
    short fNumOfRanges;
    TextRange fRange[1];
};
typedef struct TextRangeArray TextRangeArray;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AERegistry.h

TScriptingSizeResource

Defines a data type to store stack and heap information. Not typically used by developers.

```
struct TScriptingSizeResource {
    short scriptingSizeFlags;
    unsigned long minStackSize;
    unsigned long preferredStackSize;
    unsigned long maxStackSize;
    unsigned long minHeapSize;
    unsigned long preferredHeapSize;
    unsigned long maxHeapSize;
};
typedef struct TScriptingSizeResource TScriptingSizeResource;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEUserTermTypes.h

WritingCode

```
struct WritingCode {
    ScriptCode theScriptCode;
    LangCode theLangCode;
};
typedef struct WritingCode WritingCode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AERegistry.h

AEAddressDesc

A descriptor that contains the address of an application. Typically used to describe the target application for an Apple event.

```
typedef AEDesc AEAddressDesc;
```

Discussion
An address descriptor is identical to a descriptor of data type AEDesc (page 546); however, the data for an address descriptor must always consist of the address of an application.

Every Apple event includes an attribute specifying the address of the target application. The address in an address descriptor can be specified as one of these types (or as any other descriptor type you define that can be coerced to one of these types): typeApplSignature, typeSessionID, or typeProcessSerialNumber. These constants are described in "Descriptor Type Constants" (page 581). You can also use "typeApplicationBundleID" (page 628).

If your application sends Apple events to itself using a typeProcessSerialNumber address descriptor with the lowLongOfPSN field set to kCurrentProcess (and the highLongOfPSN field set to 0), the Apple Event Manager jumps directly to the appropriate Apple event handler without going through the normal event-processing sequence.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEArrayDataPointer

A pointer to a union of type AEArrayData.

```
typedef AEArrayData * AEArrayDataPointer
```

Discussion
This data type merely defines a pointer to an AEArrayData (page 545) union.

AEArrayType

Stores a value that specifies an array type.

```
typedef SInt8 AEArrayType;
```

Discussion
You use this data type with the `AEGetArray` (page 428) function and the `AEPutArray` (page 459) function to specify an array type, using one of the constants from "Data Array Constants" (page 580).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoerceDescUPP

Defines a data type for the universal procedure pointer for the `AECoerceDescProcPtr` callback function pointer.

```
typedef AECoerceDescProcPtr AECoerceDescUPP;
```

Discussion
For a description of a coerce descriptor callback function, see `AECoerceDescProcPtr` (page 524).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoercePtrUPP

Defines a data type for the universal procedure pointer for the `AECoercePtrProcPtr` callback function pointer.

```
typedef AECoercePtrProcPtr AECoercePtrUPP;
```

Discussion
For a description of a coerce pointer callback function, see `AECoercePtrProcPtr` (page 525).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoercionHandlerUPP

Defines a data type for the universal procedure pointer for the `AECoercionHandlerUPP` callback function pointer.

```
typedef AECoerceDescUPP AECoercionHandlerUPP;
```

Discussion
For a description of a coercion handler callback function, see `AECoercePtrProcPtr` (page 525).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDataStorage

A pointer to an opaque data type that provides storage for an `AEDesc` descriptor.

```
typedef AEStorageDataType * AEDataStorage;
```

Discussion
The Apple Event Manager defines the `AEDataStorage` data type to serve as a data storage field in the `AEDesc` (page 546) structure. Your application doesn't access the data pointed to by a data storage pointer directly. Rather, you work with the following functions:

- `AEGetDescDataSize` (page 434)
- `AEGetDescData` (page 432)
- `AEGetDescDataRange` (page 433)
- `AEReplaceDescData` (page 472)

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDataStorageType

An opaque data type used to store data in Apple event descriptors.

```
typedef struct OpaqueAEDataStorageType * AEDataStorageType;
```

Discussion
See `AEDesc` (page 546) for related information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDescList

A descriptor whose data consists of a list of one or more descriptors.

```
typedef AEDesc AEDescList;
```

Discussion
A descriptor list is identical to a descriptor of data type `AEDesc` (page 546) —the only difference is that the data in a descriptor list must always consist of a list of other descriptors.

Descriptor lists are a key building block of Apple events. Many Apple Event Manager functions take or return lists of descriptors in descriptor lists. For example, see the functions described in "Counting the Items in Descriptor Lists" (page 398) and "Getting Items From Descriptor Lists" (page 403).

The format of the data in the `dataHandle` of the descriptor is private. You can only operate on the contained elements with Apple Event Manager functions, including those described in "Counting the Items in Descriptor Lists" (page 398) and "Getting Items From Descriptor Lists" (page 403).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEEventSource

A data type for values that specify how an Apple event was delivered.

```
typedef SInt8 AEEventSource;
```

Discussion
"Event Source Constants" (page 588) lists the valid constant values for a variable or parameter of type `AEEventSource`.

You might use a variable of this type, for example, to get the source type of an Apple event by calling the function `AEGetAttributePtr` (page 430). You pass the `keyEventSourceAttr` constant as the value for the `theAEKeyWord` parameter and you pass a pointer to a variable of type `AEEventSource` for the `dataPtr` parameter. On return, the variable will contain one of the event source constant values described in "Event Source Constants" (page 588). The complete call looks like the following:

```
AppleEvent       theAppleEvent; // previously obtained Apple event
DescType         returnedType;
AEEventSource    sourceOfAE;
Size             actualSize;
OSErr            myErr;
myErr = AEGetAttributePtr(theAppleEvent,
                          keyEventSourceAttr,
                          typeShortInteger,
                          &returnedType,
                          (void *) &sourceOfAE,
                          sizeof (sourceOfAE),
                          &actualSize);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEDisposeExternalUPP

Defines a universal procedure pointer to a function the Apple Event Manager calls to dispose of a descriptor created by the `AECreateDescFromExternalPtr` function.

```
typedef AEDisposeExternalProcPtr AEDisposeExternalUPP;
```

Discussion
See the `AEDisposeExternalProcPtr` (page 527) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

AEEventClass

Specifies the event class of an Apple event.

```
typedef FourCharCode AEEventClass;
```

Discussion
Apple events are identified by their event class and event ID attributes. The event class is the attribute that identifies a group of related Apple events. When you call the `AEProcessAppleEvent` (page 457) function, the Apple Event Manager uses these attributes to identify a handler for a specific Apple event.

For more information on Apple event classes, see "Event Class Constants" (page 585).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEEventHandlerUPP

Defines a data type for the universal procedure pointer for the `AEEventHandlerUPP` callback function pointer.

```
typedef AEEventHandlerProcPtr AEEventHandlerUPP;
```

Discussion
For a description of an event handler callback function, see `AEEventHandlerProcPtr` (page 528).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEEventID

Specifies the event ID of an Apple event.

```
typedef FourCharCode AEEventID;
```

Discussion
Apple events are identified by their event class and event ID attributes. The event ID is the attribute that identifies a particular Apple event within its event class. In conjunction with the event class, the event ID uniquely identifies the Apple event and communicates what action the Apple event should perform.

For more information on Apple event IDs, see "Event ID Constants" (page 586).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEFilterUPP

Defines a data type for the universal procedure pointer for the `AEFilterProcPtr` callback function pointer.

```
typedef AEFilterProcPtr AEFilterUPP;
```

Discussion
For a description of a filter callback function, see `AEFilterProcPtr` (page 530).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEIdleUPP

Defines a data type for the universal procedure pointer for the `AEIdleProcPtr` callback function pointer.

```
typedef AEIdleProcPtr AEIdleUPP;
```

Discussion
For a description of an idle callback function, see `AEIdleProcPtr` (page 531).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEKeyword

A four-character code that uniquely identifies a descriptor in an Apple event record or an Apple event.

```
typedef FourCharCode AEKeyword;
```

Discussion
The Apple Event Manager uniquely identifies the various parts of an Apple event by means of keywords associated with corresponding descriptors. Keywords are arbitrary names, stored as four-character codes of type `AEKeyword`. A keyword combined with a descriptor forms a keyword-specified descriptor, which is defined by a data structure of type `AERemoteProcessResolverContext` (page 547).

The Apple Event Manager also uses keywords for Apple event attributes. Keyword constants used by the Apple Event Manager are defined in "Keyword Attribute Constants" (page 593) and "Keyword Parameter Constants" (page 595).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERecord

A descriptor whose data is a list of keyword-specified descriptors.

```
typedef AEDescList AERecord;
```

Discussion
The Apple Event Manager provides routines that allow your application to create Apple event records and extract data from them when creating or responding to Apple events. You also work with Apple event records if your application resolves or creates object specifiers. Functions that use Apple event records are described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 402) and "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 398).

The descriptor list of keyword-specified descriptors in an Apple event record must specify Apple event parameters—they cannot specify Apple event attributes. Only descriptor lists of type Apple event can contain both attributes and parameters.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERemoteProcessResolverRef

An opaque reference to an object that encapsulates the mechanism for obtaining a list of processes running on a remote machine.

```
typedef AERemoteProcessResolver * AERemoteProcessResolverRef;
```

Discussion
You create an instance of `AERemoteProcessResolverRef` by calling `AECreateRemoteProcessResolver` (page 420), and you must disposed of it by calling `AEDisposeRemoteProcessResolver` (page 424). An instance of this type is not a CFType (the base type used by all Core Foundation derived opaque types). For more information, see Core Foundation Reference Documentation.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AEReturnID

Specifies a return ID for a created Apple event.

```
typedef SInt16 AEReturnID;
```

Discussion
When you call the `AECreateAppleEvent` (page 416) function, you pass a value of type `AEReturnID` for the `returnID` parameter. "ID Constants for the AECreateAppleEvent Function" (page 589) lists the valid constant values for a variable or parameter of this type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESendOptions

This data type is not available. (**Deprecated.** Not available in Apple Event Manager API.)

```
typedef OptionBits AESendOptions;
```

AESendPriority

Specifies the processing priority for a sent Apple event.

```
typedef SInt16 AESendPriority;
```

Discussion
When you call the `AESend` (page 476) function, you pass a value of type `AESendPriority` for the `sendPriority` parameter. "Priority Constants for the AESend Function (Deprecated in Mac OS X)" (page 601) lists the valid constant values for a variable or parameter of this type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEStreamRef

An opaque data structure for storing stream-based descriptor data.

```
typedef struct OpaqueAEStreamRef * AEStreamRef;
```

Discussion
You create `AEStreamRef` objects and manipulate their contents using the "AEStream" routines found in the section "Creating Apple Event Structures Using Streams" (page 406)

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AETransactionID

Specifies a transaction ID.

```
typedef SInt32 AETransactionID;
```

Discussion
A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. When you call the `AECreateAppleEvent` (page 416) function, you pass a value of type `AETransactionID` for the `transactionID` parameter. "ID Constants for the AECreateAppleEvent Function" (page 589) lists the valid constant values for a variable or parameter of this type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AppleEvent

A descriptor whose data is a list of descriptors containing both attributes and parameters that make up an Apple event.

```
typedef AERecord AppleEvent;
```

Discussion
The Apple event data type describes a full-fledged Apple event. Like the data for an Apple event record (data type `AERecord` (page 557)), the data for an Apple event consists of a list of keyword-specified descriptors. Unlike an Apple event record, the data for an Apple event is conceptually divided into two parts, one for attributes and one for parameters. This division within the Apple event allows the Apple Event Manager to distinguish between an event's attributes and its parameters.

For additional information on the structure of an Apple event and on how to build one, see "Building an Apple Event" in Apple Events Programming Guide.

Many functions work with Apple events, including the functions described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 402), "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 398), "Creating an Apple Event" (page 399), and "Sending an Apple Event" (page 405).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DescType

Specifies the type of the data stored in an `AEDesc` descriptor.

```
typedef ResType DescType;
```

Discussion
A `DescType` data type is a four-character code that stores a value that identifies the data in an `AEDesc` (page 546) descriptor, the basic building block for all Apple events.

The descriptor type constants used by the Apple Event Manager are described in "Descriptor Type Constants" (page 581) and "Key Form and Descriptor Type Object Specifier Constants" (page 590).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

OffsetArrayHandle

Defines a data type that points to an `OffsetArray`. Not typically used by developers.

```
typedef OffsetArrayPtr * OffsetArrayHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AERegistry.h`

OSLAccessorUPP

Defines a data type for the universal procedure pointer for the `OSLAccessorProcPtr` callback function pointer.

```
typedef OSLAccessorProcPtr OSLAccessorUPP;
```

Discussion
For a description of an object accessor callback function, see `OSLAccessorProcPtr` (page 533).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLAdjustMarksUPP

Defines a data type for the universal procedure pointer for the `OSLAdjustMarksProcPtr` callback function pointer.

```
typedef OSLAdjustMarksProcPtr OSLAdjustMarksUPP;
```

Discussion
For a description of an adjust marks callback function, see `OSLAdjustMarksProcPtr` (page 535).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCompareUPP

Defines a data type for the universal procedure pointer for the `OSLCompareProcPtr` callback function pointer.

```
typedef OSLCompareProcPtr OSLCompareUPP;
```

Discussion
For a description of a compare callback function, see `OSLCompareProcPtr` (page 536).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCountUPP

Defines a data type for the universal procedure pointer for the `OSLCountProcPtr` callback function pointer.

```
typedef OSLCountProcPtr OSLCountUPP;
```

Discussion
For a description of a count callback function, see `OSLCountProcPtr` (page 538).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLDisposeTokenUPP

Defines a data type for the universal procedure pointer for the `OSLDisposeTokenProcPtr` callback function pointer.

```
typedef OSLDisposeTokenProcPtr OSLDisposeTokenUPP;
```

Discussion
For a description of a dispose token callback function, see `OSLDisposeTokenProcPtr` (page 539).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetErrDescUPP

Defines a data type for the universal procedure pointer for the `OSLGetErrDescProcPtr` callback function pointer.

```
typedef OSLGetErrDescProcPtr OSLGetErrDescUPP;
```

Discussion
For a description of a get error descriptor callback function, see `OSLGetErrDescProcPtr` (page 541).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetMarkTokenUPP

Defines a data type for the universal procedure pointer for the `OSLGetMarkTokenProcPtr` callback function pointer.

```
typedef OSLGetMarkTokenProcPtr OSLGetMarkTokenUPP;
```

Discussion
For a description of a mark token callback function, see `OSLGetMarkTokenProcPtr` (page 542).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLMarkUPP

Defines a data type for the universal procedure pointer for the `OSLMarkProcPtr` callback function pointer.

```
typedef OSLMarkProcPtr OSLMarkUPP;
```

Discussion
For a description of a mark callback function, see `OSLMarkProcPtr` (page 544).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEInteractAllowed

Specifies an interaction level.

```
typedef SInt8 AEInteractAllowed;
```

Discussion
When you call the `AEGetInteractionAllowed` (page 436) function or the `AESetInteractionAllowed` (page 479) function, you receive or pass a value of type `AEInteractAllowed` for the `level` parameter. Interaction levels are described and the valid interaction level constants are listed in "User Interaction Level Constants" (page 605).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

Constants

AEBuild Error Codes

Represents syntax errors found by an "AEBuild" routine.

```
typedef UInt32 AEBuildErrorCode;
enum {
aeBuildSyntaxNoErr = 0,
aeBuildSyntaxBadToken = 1,
aeBuildSyntaxBadEOF = 2,
aeBuildSyntaxNoEOF = 3,
aeBuildSyntaxBadNegative = 4,
aeBuildSyntaxMissingQuote = 5,
aeBuildSyntaxBadHex = 6,
aeBuildSyntaxOddHex = 7,
aeBuildSyntaxNoCloseHex = 8,
aeBuildSyntaxUncoercedHex = 9,
aeBuildSyntaxNoCloseString = 10,
aeBuildSyntaxBadDesc = 11,
aeBuildSyntaxBadData = 12,
aeBuildSyntaxNoCloseParen = 13,
aeBuildSyntaxNoCloseBracket = 14,
aeBuildSyntaxNoCloseBrace = 15,
aeBuildSyntaxNoKey = 16,
aeBuildSyntaxNoColon = 17,
aeBuildSyntaxCoercedList = 18,
aeBuildSyntaxUncoercedDoubleAt = 19
};
```

Constants

`aeBuildSyntaxNoErr`
No error.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadToken`
An illegal character was specified.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadEOF`
An unexpected end of format string was encountered.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxNoEOF`
There were unexpected characters beyond the end of the format string.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadNegative`
A minus sign “-” was not followed by digits.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxMissingQuote`
A string was not terminated by a closing quotation mark.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadHex`
A hex string contained characters other than hexadecimal digits.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxOddHex`
A hex string contained an odd number of digits.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseHex`
A hex string was missing a "$" or "»" character.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxUncoercedHex`
A hex string must be coerced to a type.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseString`
A string was missing a closing quote.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxBadDesc`
An illegal descriptor was specified.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxBadData`
Bad data was found inside a variable argument list.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseParen`
A data value was missing a closing parenthesis.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseBracket`
A comma or closing bracket "]" was expected.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseBrace`
A comma or closing brace "}" was expected.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoKey`
A keyword was missing from a descriptor.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxNoColon`
In a descriptor, one of the keywords was not followed by a colon.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxCoercedList`
Cannot coerce a list.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxUncoercedDoubleAt`
You must coerce a "@@" substitution.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

AESendMode

Specify send preferences to the `AESend` function.

```
typedef SInt32 AESendMode;
enum {
    kAENoReply = 0x00000001,
    kAEQueueReply = 0x00000002,
    kAEWaitReply = 0x00000003,
    kAEDontReconnect = 0x00000080,
    kAEWantReceipt = 0x00000200,
    kAENeverInteract = 0x00000010,
    kAECanInteract = 0x00000020,
    kAEAlwaysInteract = 0x00000030,
    kAECanSwitchLayer = 0x00000040,
    kAEDontRecord = 0x00001000,
    kAEDontExecute = 0x00002000,
    kAEProcessNonReplyEvents = 0x00008000
};
```

Constants

`kAENoReply`
The reply preference—your application does not want a reply Apple event. If you set the bit specified by this constant, the server processes the Apple event as soon as it has the opportunity.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEQueueReply`

The reply preference—your application wants a reply Apple event. If you set the bit specified by this constant, the reply appears in your event queue as soon as the server has the opportunity to process and respond to your Apple event.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEWaitReply`

The reply preference—your application wants a reply Apple event and is willing to give up the processor while waiting for the reply. For example, if the server application is on the same computer as your application, your application yields the processor to allow the server to respond to your Apple event.

If you set the bit specified by this constant, you must provide an idle function. This function should process any update events, null events, operating-system events, or activate events that occur while your application is waiting for a reply. For more information on idle routines, see `AEInteractWithUser` (page 453).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDontReconnect`

Deprecated and unsupported in Mac OS X. The reconnection preference—the Apple Event Manager must not automatically try to reconnect if it receives a `sessClosedErr` result code from the PPC Toolbox. If you don't set this flag, the Apple Event Manager automatically attempts to reconnect and reestablish the session.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEWantReceipt`

Deprecated and unsupported in Mac OS X. The return receipt preference—the sender wants to receive a return receipt for this Apple event from the Event Manager. (A return receipt means only that the receiving application accepted the Apple event the Apple event may or may not be handled successfully after it is accepted.) If the receiving application does not send a return receipt before the request times out, `AESend` returns `errAETimeout` as its function result.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAENeverInteract`

The user interaction preference—the server application should never interact with the user in response to the Apple event. If you set the bit specified by this constant, the `AEInteractWithUser` (page 453) function (when called by the server) returns the `errAENoUserInteraction` result code. When you send an Apple event to a remote application, the default is to set this bit.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAECanInteract`

The user interaction preference—the server application can interact with the user in response to the Apple event. By convention, you set the bit specified by this constant if the user needs to supply information to the server. If you set the bit and the server allows interaction, the `AEInteractWithUser` (page 453) function either brings the server application to the foreground or posts a notification request. When you send an Apple event to a local application, the default is to set this bit.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEAlwaysInteract`

The user interaction preference—the server application should always interact with the user in response to the Apple event. By convention, you set the bit specified by this constant whenever the server application normally asks a user to confirm a decision or interact in any other way, even if no additional information is needed from the user. If you set the bit specified by this constant, the `AEInteractWithUser` (page 453) function either brings the server application to the foreground or posts a notification request.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAECanSwitchLayer`

The application switch preference—if both the client and server allow interaction, and if the client application is the active application on the local computer and is waiting for a reply (that is, it has set the `kAEWaitReply` flag), `AEInteractWithUser` brings the server directly to the foreground. Otherwise, `AEInteractWithUser` uses the Notification Manager to request that the user bring the server application to the foreground.

You should specify the `kAECanSwitchLayer` flag only when the client and server applications reside on the same computer. In general, you should not set this flag if it would be confusing or inconvenient to the user for the server application to come to the front unexpectedly. This flag is ignored if you are sending an Apple event to a remote computer.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDontRecord`

The recording preference—your application is sending an event to itself but does not want the event recorded. When Apple event recording is on, the Apple Event Manager records a copy of every event your application sends to itself except for those events for which this flag is set.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDontExecute`

The execution preference—your application is sending an Apple event to itself for recording purposes only—that is, you want the Apple Event Manager to send a copy of the event to the recording process but you do not want your application actually to receive the event.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEProcessNonReplyEvents`

Allow processing of non-reply Apple events while awaiting a synchronous Apple event reply (you specified `kAEWaitReply` for the reply preference).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
You use these constants with the `sendMode` parameter to the `AESend` (page 476) function to specify how the server application should handle the reply mode, the interaction level, the application switch mode, the reconnection mode, the return receipt mode, the recording mode, and whether to process non-reply Apple events. To obtain a value for this parameter, you add together constants to set the appropriate bits for the Apple event you are about to send. The following paragraphs provide additional information about how you use these constants.

You can set only one flag reply preference (`kAENoReply`, `kAEQueueReply`, or `kAEWaitReply`), one user interaction preference (`kAENeverInteract`, `kAECanInteract`, or `kAEAlwaysInteract`), and one recording and execution preference (`kAEDontRecord` or `kAEDontExecute`).

Before the Apple Event Manager sends a reply event back to the client application, the `keyAddressAttr` attribute contains the address of the client application. After the client receives the reply event, the `keyAddressAttr` attribute contains the address of the server application.

If you specify `kAEWaitReply`, the Apple Event Manager uses the Event Manager to send the event. The Apple Event Manager then calls the `WaitNextEvent` function on behalf of your application, causing your application to yield the processor and giving the server application a chance to receive and handle the Apple event. Your application continues to yield the processor until the server handles the Apple event or the request times out.

Specify the `kAEWantReceipt` flag if your application wants notification that the server application has accepted the Apple event. If you specify this flag, your application receives a return receipt as a high-level event.

If you specify the `kAEWantReceipt` flag and the server application does not accept the Apple event within the time specified by the `timeOutInTicks` parameter to `AESend`, the `AESend` function returns a timeout error. Note that `AESend` also returns a timeout error if your application sets the `kAEWaitReply` flag and does not receive the reply Apple event within the time specified by the `timeOutInTicks` parameter.

You use one of the three flags—`kAENeverInteract`, `kAECanInteract`, and `kAEAlwaysInteract`—to specify whether the server should interact with the user when handling the Apple event. Specify `kAENeverInteract` if the server should not interact with the user when handling the Apple event. You might specify this constant if you don't want the user to be interrupted while the server is handling the Apple event.

Use the `kAECanInteract` flag if the server should interact with the user when the user needs to supply information to the server. Use the `kAEAlwaysInteract` flag if the server should interact with the user whenever the server normally asks a user to confirm a decision or interact in any other way, even if no additional information is needed from the user. Note that it is the responsibility of the server and client applications to agree on how to interpret the `kAEAlwaysInteract` flag.

If the client application does not set any one of the user interaction flags, the Apple Event Manager sets a default, depending on the location of the target of the Apple event. If the server application is on a remote computer, the Apple Event Manager sets the `kAENeverInteract` flag as the default. If the target of the Apple event is on the local computer, the Apple Event Manager sets the `kAECanInteract` flag as the default.

The server application should call `AEInteractWithUser` if it needs to interact with the user. If both the client and the server allow user interaction, the Apple Event Manager attempts to bring the server to the foreground if it is not already the foreground process. If both the `kAECanSwitchLayer` and the `kAEWaitReply` flags are set, and if the client application is the active application on the local computer, the Apple Event Manager brings the server application directly to the front. Otherwise, the Apple Event Manager

posts a notification request asking the user to bring the server application to the front, regardless of whether the `kAECanSwitchLayer` flag is set. This ensures that the user will not be interrupted by an unexpected application switch.

Specify the `kAEDontRecord` flag if your application is sending an Apple event to itself that you don't want to be recorded. When Apple event recording has been turned on, every event that your application sends to itself will be automatically recorded by the Apple Event Manager except those sent with the `kAEDontRecord` flag set.

Specify the `kAEDontExecute` flag if your application is sending an Apple event to itself for recording purposes only—that is, if you want the Apple Event Manager to send a copy of the event to the recording process but you do not want your application actually to receive the event.

See also "Requesting User Interaction" (page 405).

Version Notes
The `kAEDontReconnect` and `kAEWantReceipt` constants are deprecated and unsupported in Mac OS X.

Declared In
`AEDataModel.h`

Apple Event Recording Event ID Constants

Specify event IDs for events that deal with Apple event recording.

```
enum {
    kAEStartRecording = 'reca',
    kAEStopRecording = 'recc',
    kAENotifyStartRecording = 'rec1',
    kAENotifyStopRecording = 'rec0',
    kAENotifyRecording = 'recr'
};
```

Constants

`kAEStartRecording`

Event ID for an event by a scripting component to the recording process (or to any running process on the local computer), but handled by the Apple Event Manager. The Apple Event Manager responds by turning on recording and sending a `recording on` event to all running processes on the local computer.

If sent by process serial number (PSN), this event must be addressed using a real PSN; it should never be sent to an address specified as `kCurrentProcess`.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEStopRecording`

Event ID for an event sent by a scripting component to the recording process (or to any running process on the local computer), but handled by the Apple Event Manager. The Apple Event Manager responds by sending a `recording off` event to all running processes on the local computer.

If sent by a PSN, this event must be addressed using a real PSN; it should never be sent to an address specified as `kCurrentProcess`.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAENotifyStartRecording`
: An event that notifies an application that recording has been turned on.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAENotifyStopRecording`
: An event that notifies an application that recording has been turned off.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAENotifyRecording`
: Wildcard event class and event ID handled by a recording process in order to receive and record copies of recordable events sent to it by the Apple Event Manager. Scripting components install a handler for this event on behalf of a recording process when recording is turned on and remove the handler when recording is turned off.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

Version Notes
These constants are available only in version 1.0.1 and later of the Apple Event Manager.

cAEList

```
enum {
    cAEList = 'list',
    cApplication = 'capp',
    cArc = 'carc',
    cBoolean = 'bool',
    cCell = 'ccel',
    cChar = 'cha ',
    cColorTable = 'clrt',
    cColumn = 'ccol',
    cDocument = 'docu',
    cDrawingArea = 'cdrw',
    cEnumeration = 'enum',
    cFile = 'file',
    cFixed = 'fixd',
    cFixedPoint = 'fpnt',
    cFixedRectangle = 'frct',
    cGraphicLine = 'glin',
    cGraphicObject = 'cgob',
    cGraphicShape = 'cgsh',
    cGraphicText = 'cgtx',
    cGroupedGraphic = 'cpic'
};
```

Callback Constants for the AEResolve Function

Specify supported callback features to the `AEResolve` function.

```
enum {
    kAEIDoMinimum = 0x0000,
    kAEIDoWhose = 0x0001,
    kAEIDoMarking = 0x0004,
    kAEPassSubDescs = 0x0008,
    kAEResolveNestedLists = 0x0010,
    kAEHandleSimpleRanges = 0x0020,
    kAEUseRelativeIterators = 0x0040
};
```

Constants

`kAEIDoMinimum`

The application does not handle whose tests or provide marking callbacks.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`kAEIDoWhose`

The application supports whose tests (supports key form `formWhose`).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`kAEIDoMarking`

The application provides marking callback functions. Marking callback functions are described in "Object Callback Functions" (page 524).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Discussion

You use these constants to supply a value for the `callbackFlags` parameter to the `AEResolve` (page 473) function. This value specifies whether your application supports whose descriptors or provides marking callback functions. To obtain a value for this parameter, you can add together constants to set the appropriate bits, as shown in the following example (for an application that supports both whose tests and marking):

```
    AEDesc objectSpecifier; // Previously obtained object specifier.     AEDesc
  resultToken;
    OSErr myErr;

    myErr = AEResolve (&objectSpecifier,
                       kAEIDoWhose + kAEIDoMarking, &resultToken)
```

AppleScript generates whose clauses from script statements such as the following:

```
tell application "Finder"
    every file in control panels folder whose file type is "APPL"
end tell
```

cInsertionLoc

```
enum {
    cInsertionLoc = 'insl',
    cInsertionPoint = 'cins',
    cIntlText = 'itxt',
    cIntlWritingCode = 'intl',
    cItem = 'citm',
    cLine = 'clin',
    cLongDateTime = 'ldt ',
    cLongFixed = 'lfxd',
    cLongFixedPoint = 'lfpt',
    cLongFixedRectangle = 'lfrc',
    cLongInteger = 'long',
    cLongPoint = 'lpnt',
    cLongRectangle = 'lrct',
    cMachineLoc = 'mLoc',
    cMenu = 'cmnu',
    cMenuItem = 'cmen',
    cObject = 'cobj',
    cObjectSpecifier = 'obj ',
    cOpenableObject = 'coob',
    cOval = 'covl'
};
```

cKeystroke

```
enum {
    cKeystroke = 'kprs',
    pKeystrokeKey = 'kMsg',
    pModifiers = 'kMod',
    pKeyKind = 'kknd',
    eModifiers = 'eMds',
    eOptionDown = 'Kopt',
   eCommandDown = 'Kcmd',
    eControlDown = 'Kctl',
    eShiftDown = 'Ksft',
    eCapsLockDown = 'Kclk',
    eKeyKind = 'ekst',
    eEscapeKey = 0x6B733500,
    eDeleteKey = 0x6B733300,
    eTabKey = 0x6B733000,
    eReturnKey = 0x6B732400,
    eClearKey = 0x6B734700,
    eEnterKey = 0x6B734C00,
    eUpArrowKey = 0x6B737E00,
    eDownArrowKey = 0x6B737D00,
    eLeftArrowKey = 0x6B737B00,
    eRightArrowKey = 0x6B737C00,
    eHelpKey = 0x6B737200,
    eHomeKey = 0x6B737300,
    ePageUpKey = 0x6B737400,
    ePageDownKey = 0x6B737900,
    eForwardDelKey = 0x6B737500,
    eEndKey = 0x6B737700,
    eF1Key = 0x6B737A00,
```

```
    eF2Key = 0x6B737800,
    eF3Key = 0x6B736300,
    eF4Key = 0x6B737600,
    eF5Key = 0x6B736000,
    eF6Key = 0x6B736100,
    eF7Key = 0x6B736200,
    eF8Key = 0x6B736400,
    eF9Key = 0x6B736500,
    eF10Key = 0x6B736D00,
    eF11Key = 0x6B736700,
    eF12Key = 0x6B736F00,
    eF13Key = 0x6B736900,
    eF14Key = 0x6B736B00,
    eF15Key = 0x6B737100
};
```

Comparison Operator Constants

Specify a comparison operation to perform on two operands.

```
enum {
    kAEAsk = 'ask ',
    kAEBefore = 'befo',
    kAEBeginning = 'bgng',
    kAEBeginsWith = 'bgwt',
    kAEBeginTransaction = 'begi',
    kAEBold = 'bold',
    kAECaseSensEquals = 'cseq',
    kAECentered = 'cent',
    kAEChangeView = 'view',
    kAEClone = 'clon',
    kAEClose = 'clos',
    kAECondensed = 'cond',
    kAEContains = 'cont',
    kAECopy = 'copy',
    kAECoreSuite = 'core',
    kAECountElements = 'cnte',
    kAECreateElement = 'crel',
    kAECreatePublisher = 'cpub',
    kAECut = 'cut ',
    kAEDelete = 'delo'
};
```

Constants

`kAEBeginsWith`

The value of `operand1` begins with the value of `operand2` (for example, the string `"operand"` begins with the string `"opera"`).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEContains`

The value of `operand1` contains the value of `operand2` (for example, the string `"operand"` contains the string `"era"`).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAECoreSuite`
: An Apple event in the Standard Suite.

  Available in Mac OS X v10.0 and later.

  Declared in `AERegistry.h`.

Discussion
When you call the `CreateCompDescriptor` (page 498) function, you pass one of these comparison operators in the `comparisonOperator` parameter. The `CreateCompDescriptor` function creates a comparison descriptor that specifies how to compare one or more Apple event objects with either another Apple event object or a descriptor.

The actual comparison of the two operands is performed by the object comparison function provided by the client application—see `OSLCompareProcPtr` (page 536). The way a comparison operator is interpreted is up to each application.

For related information, see "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 575).

Constants for Object Specifiers, Positions, and Logical and Comparison Operations

Specify the types of the four keyword-specified descriptors that make up the data in an object specifier, as well as constants for position, logical operations, and comparison operations.

```
enum {
    kAEAND = 'AND ',
    kAEOR = 'OR  ',
    kAENOT = 'NOT ',
    kAEFirst = 'firs',
    kAELast = 'last',
    kAEMiddle = 'midd',
    kAEAny = 'any ',
    kAEAll = 'all ',
    kAENext = 'next',
    kAEPrevious = 'prev',
    keyAECompOperator = 'relo',
    keyAELogicalTerms = 'term',
    keyAELogicalOperator = 'logc',
    keyAEObject1 = 'obj1',
    keyAEObject2 = 'obj2',
    keyAEDesiredClass = 'want',
    keyAEContainer = 'from',
    keyAEKeyForm = 'form',
    keyAEKeyData = 'seld'
};
```

Constants
`kAEAND`
: Specifies a logical `AND` operation.

  Available in Mac OS X v10.0 and later.

  Declared in `AEObjects.h`.

`kAEOR`
Specifies a logical `OR` operation.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAENOT`
Specifies a logical `NOT` operation.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEFirst`
The first element in the specified container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAELast`
Specifies the last element in the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEMiddle`
Specifies the middle element in the container. If an object specifier specifies `kAEMiddle` and the number of elements in the container is even, the Apple Event Manager rounds down. For example, in a range of four words the second word is the "middle" word.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEAny`
Specifies a single element chosen at random from the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEAll`
Specifies all the elements in the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAENext`
Specifies the Apple event object after the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEPrevious`
Specifies the Apple event object before the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAECompOperator`
Specifies a descriptor of `typeType`, whose data consists of one of the constant values described in "Key Form and Descriptor Type Object Specifier Constants" (page 590).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAELogicalTerms`
Specifies a descriptor of type `typeAEList` containing one or more comparison or logical descriptors.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAELogicalOperator`
Specifies a descriptor of type `typeEnumerated` whose data is one of the logical operators (such as `kAEAND`) defined in "Key Form and Descriptor Type Object Specifier Constants" (page 590).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEObject1`
Identifies a descriptor for the element that is currently being compared to the object or data specified by the descriptor for the keyword `keyAEObject2`. Either object can be described by a descriptor of type `typeObjectSpecifier` or `typeObjectBeingExamined`.

A descriptor of `typeObjectBeingExamined` acts as a placeholder for each of the successive elements in a container when the Apple Event Manager tests those elements one at a time.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEObject2`
Identifies a descriptor for the element that is currently being compared to the object or data specified by the descriptor for the keyword `keyAEObject1`.

The keyword `keyAEObject2` can also be used with a descriptor of any other descriptor type whose data is to be compared to each element in a container.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEDesiredClass`
A four-character code that identifies the object class of the specified object or objects.

Constants for object class IDs are described in "Key Form and Descriptor Type Object Specifier Constants" (page 590).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEContainer`
Specifies the container for the requested object or objects. The data is an object specifier (or in some cases a null descriptor).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEKeyForm`
A four-character code that identifies the key form for the specified object or objects.

The constants for specifying the key form are described in "Key Form and Descriptor Type Object Specifier Constants" (page 590).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEKeyData`

Data or nested descriptors that specify a property, name, position, range, or test, depending on the key form.

The descriptor types used in object specifiers are described in "Key Form and Descriptor Type Object Specifier Constants" (page 590).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Discussion

When you call the `CreateLogicalDescriptor` (page 499) function to create a logical descriptor, you pass one of the logical operators `kAEAND`, `kAEOR`, or `kAENOT` in the `theLogicOperator` parameter. The `CreateLogicalDescriptor` function creates a logical descriptor that specifies a logical operation to perform on one or more operands.

The constants `kAEFirst`, `kAELast`, `kAEMiddle`, `kAEAny`, and `kAEAll` provide the key data for a keyword-specified descriptor of key form `formAbsolutePosition` and descriptor type `typeAbsoluteOrdinal`.

The constants `kAENext`, and `kAEPrevious` provide the key data for a keyword-specified descriptor of key form `formRelativePosition`.

Key form constants and descriptor type constants for object specifiers are defined in "Key Form and Descriptor Type Object Specifier Constants" (page 590).

The constants `keyAELogicalTerms` and `keyAELogicalOperator` define the keyword descriptors for a logical descriptor. A logical descriptor is a coerced Apple event record of type `typeLogicalDescriptor` that specifies a logical expression—that is, an expression that the Apple Event Manager evaluates to either `TRUE` or `FALSE`. You can create a logical descriptor with the `CreateLogicalDescriptor` (page 499) function.

The data for a logical descriptor consists of two keyword-specified descriptors: the first with descriptor type `keyAELogicalOperator`, descriptor type `typeEnumerated`, and one of the logical operators defined in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 575) for its data; and the second with descriptor type `keyAELogicalTerms`, descriptor type `typeEnumerated`, and one or more comparison or logical descriptors for its data. Comparison constants are described in "Comparison Operator Constants" (page 574).

The logical expression is constructed from a logical operator (one of the Boolean operators `AND`, `OR`, or `NOT`) and a list of logical terms to which the operator is applied (where `NOT` can only be used where the list of terms is a single-item list). Each logical term in the list can be either another logical descriptor or a comparison descriptor (described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 575)).

The Apple Event Manager short-circuits its evaluation of a logical expression as soon as one part of the expression fails a test. For example, if while testing a logical expression such as A `AND` B `AND` C the Apple Event Manager discovers that A `AND` B is not true, it will evaluate the expression to `FALSE` without testing C.

The constants `keyAECompOperator`, `keyAEObject1`, and `keyAEObject2` define the keyword descriptors for a comparison descriptor. A comparison descriptor is a coerced Apple event record of type `typeCompDescriptor` that specifies an Apple event object and either another Apple event object or data for the Apple Event Manager to compare to the first object. You can create a logical descriptor with the `CreateCompDescriptor` (page 498) function.

The Apple Event Manager can also use the information in a comparison descriptor to compare elements in a container, one at a time, either to an Apple event object or to data. The data for a comparison descriptor consists of three keyword-specified descriptors:

- A descriptor with keyword `keyAECompOperator`, descriptor type `typeType`, and one of the logical operators defined in "Comparison Operator Constants" (page 574) for its data.
- A descriptor with keyword `keyAEObject1` and either
- descriptor type `typeObjectSpecifier` and object specifier data to compare, or
- descriptor type `typeObjectBeingExamined` and a data storage pointer of `NULL`.
- A descriptor with keyword `keyAEObject2` and either
- descriptor type `typeObjectSpecifier` and object specifier data to compare, or
- descriptor type `typeObjectBeingExamined` and a data storage pointer of `NULL`, or
- any other descriptor type and the data to be compared for that descriptor type.

You don't have to support all the available comparison operators for all Apple event objects for example, the `begins with` operator probably doesn't make sense for objects of type `cRectangle`. It is up to you to decide which comparison operators are appropriate for your application to support, and how to interpret them. If necessary, you can define your own custom comparison operators. If you think you need to do this, check the Apple Events and Scripting header files to see if existing definitions of comparison operators can be adapted to the needs of your application.

An object specifier is a coerced Apple event record of descriptor type `typeObjectSpecifier` whose data contains consists of four keyword-specified descriptors. The constants `keyAEDesiredClass`, `keyAEContainer`, `keyAEKeyForm`, and `keyAEKeyData` specify the keywords for the four descriptor types that together identify the specified object or objects.

cURL

```
enum {
    cURL = 'url ',
    cInternetAddress = 'IPAD',
    cHTML = 'html',
    cFTPItem = 'ftp '
};
```

Constants

`cURL`

Specifies a Uniform Resource Locator or Uniform Resource ID (URI).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cInternetAddress`

Specifies an Internet or Intranet address for the TCP/IP protocol.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cHTML`
Specifies HTML (HyperText Markup Language) format.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cFTPItem`
Specifies FTP (File Transfer Protocol) protocol.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

cVersion

```
enum {
    cVersion = 'vers',
    cWindow = 'cwin',
    cWord = 'cwor',
    enumArrows = 'arro',
    enumJustification = 'just',
    enumKeyForm = 'kfrm',
    enumPosition = 'posi',
    enumProtection = 'prtn',
    enumQuality = 'qual',
    enumSaveOptions = 'savo',
    enumStyle = 'styl',
    enumTransferMode = 'tran',
    formUniqueID = 'ID  ',
    kAEAbout = 'abou',
    kAEAfter = 'afte',
    kAEAliasSelection = 'sali',
    kAEAllCaps = 'alcp',
    kAEArrowAtEnd = 'aren',
    kAEArrowAtStart = 'arst',
    kAEArrowBothEnds = 'arbo'
};
```

Constants

`formUniqueID`
Specifies a value that uniquely identifies an object within its container or across an application.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Data Array Constants

Specify an array type for storing or extracting descriptor lists with the `AEPutArray` and `AEGetArray` functions.

```
enum {
    kAEDataArray = 0,
    kAEPackedArray = 1,
    kAEDescArray = 3,
    kAEKeyDescArray = 4
};
```

Constants

`kAEDataArray`

Array items consist of data of the same size and same type, and are aligned on word boundaries.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEPackedArray`

Array items consist of data of the same size and same type, and are packed without regard for word boundaries.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDescArray`

Array items consist of descriptors of different descriptor types with data of variable size.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEKeyDescArray`

Array items consist of keyword-specified descriptors with different keywords, different descriptor types, and data of variable size.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

When your application calls the `AEPutArray` (page 459) function to put information into a descriptor list or the `AEGetArray` (page 428) function to get information from a descriptor list, it uses an array to store the information. The type of array depends on the data for the array, as specified by one of these constants.

Array items in Apple event arrays of type `kAEDataArray`, `kAEPackedArray`, or `kAEHandleArray` must be factored—that is, contained in a factored descriptor list. For more information, see `AEPutArray` (page 459).

Descriptor Type Constants

Specify types for descriptors.

```
enum {
    typeAEList = 'list',
    typeAERecord = 'reco',
    typeAppleEvent = 'aevt',
    typeEventRecord = 'evrc',
    typeTrue = 'true',
    typeFalse = 'fals',
    typeAlias = 'alis',
    typeEnumerated = 'enum',
    typeType = 'type',
    typeAppParameters = 'appa',
    typeProperty = 'prop',
    typeFSS = 'fss ',
    typeFSRef = 'fsrf',
    typeFileURL = 'furl',
    typeKeyword = 'keyw',
    typeSectionH = 'sect',
    typeWildCard = '****',
    typeApplSignature = 'sign',
    typeQDRectangle = 'qdrt',
    typeFixed = 'fixd',
    typeProcessSerialNumber = 'psn ',
    typeApplicationURL = 'aprl',
    typeNull = 'null'
};
```

Constants

`typeAEList`
List of descriptors.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeAERecord`
List of keyword-specified descriptors.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeAppleEvent`
Apple event.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeTrue`
`TRUE` Boolean value.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeFalse`
`FALSE` Boolean value.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeAlias`
Alias.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeEnumerated`
Enumerated data.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeType`
Four-character code for event class or event ID

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeAppParameters`
Process Manager launch parameters.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeProperty`
Apple event object property.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeFSS`
File system specification. Deprecated in Mac OS X. Use file system references (`typeFSRef`) instead.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeFSRef`
File system reference. Use in preference to file system specifications (`typeFSS`).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeFileURL`
A file URL. That is, the associated data consists of the bytes of a UTF-8 encoded URL with a scheme of "file". This type is appropriate for describing a file that may not yet exist—see Technical Note 2022 for more information.

You can translate between a descriptor of this type and an instance of `CFURL` by calling `CFURLCreateWithBytes` and specifying `kCFStringEncodingUTF8` for the encoding. Or, if you have a `CFURLRef`, you can call `CFURLCreateData` to get the data as an instance of `CFData` (again specifying an encoding of `kCFStringEncodingUTF8`), and`CFDataGetBytes` to get the actual bytes to insert into a descriptor of this type.

Available in Mac OS X v10.1 and later.

Declared in `AEDataModel.h`.

`typeKeyword`
Apple event keyword.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeSectionH`
: Handle to a section record. (Deprecated.)

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeWildCard`
: Matches any type.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeApplSignature`
: Application signature.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeProcessSerialNumber`
: A process serial number. See also `AEAddressDesc` (page 551).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeApplicationURL`
: For specifying an application by URL. See Discussion section below for important information.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeNull`
: A null data storage pointer. When resolving an object specifier, an object with a null storage pointer specifies the default container at the top of the container hierarchy.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
The constants described here specify the data type for a descriptor and show the kind of data stored in a descriptor with that type.

Descriptors are the building blocks used by the Apple Event Manager to construct Apple event attributes and parameters. A descriptor is a data structure of type `AEDesc` (page 546), which consists of data storage and a descriptor type that identifies the type of the data. A descriptor type is defined by the data type `DescType` (page 560). AppleScript defines descriptor type constants for a wide variety of common data types. For additional types, see "Numeric Descriptor Type Constants" (page 597) and "Other Descriptor Type Constants" (page 601). For a complete listing, including data types such as units of length, weight, and volume, see the Apple Event Manager and Open Scripting Architecture header files.

For the constant `typeApplicationURL`, the data that specifies the application URL takes the following format:

```
eppc://[username[:password]@]host/AppName[[?uid=#]&[pid=#]]
```

As indicated by this format:

- `username` is optional. If present, an `'@'` must appear before the host name. `password` is optional. If present, `username` is not optional, and the password must be separated from the username by a `':'` and must precede the `'@'`. `AppName` is not optional; if it contains non-UTF-8 characters or white space, it must be URL-encoded (for example, `My%20Application`).

- `uid` and `pid` are optional. If `pid` is present, `uid` and `AppName` are ignored and the event is delivered only to applications with the given process id. If `uid` is present, events are directed to the application name owned by the given user id.

The following are examples of valid URLs:

```
eppc://Steve%20Zellers:wombat@grrr.apple.com/Microsoft%20Word
eppc://Steve%20Zellers:wombat@grrr.apple.com/Microsoft%20Word?pid=1284
```

The availability of user identifiers provides enhanced Apple event support for Fast User Switching. Such identifiers make it possible to send Apple events to applications running in any session, if the uids of the processes match. `'root'` (or `uid 0`) processes are allowed to send Apple events to any process in any session. Non-root processes can only target applications that match their uid.

eScheme

```
enum {
    eScheme = 'esch',
    eurlHTTP = 'http',
    eurlHTTPS = 'htps',
    eurlFTP = 'ftp ',
    eurlMail = 'mail',
    eurlFile = 'file',
    eurlGopher = 'gphr',
    eurlTelnet = 'tlnt',
    eurlNews = 'news',
    eurlSNews = 'snws',
    eurlNNTP = 'nntp',
    eurlMessage = 'mess',
    eurlMailbox = 'mbox',
    eurlMulti = 'mult',
    eurlLaunch = 'laun',
    eurlAFP = 'afp ',
    eurlAT = 'at  ',
    eurlEPPC = 'eppc',
    eurlRTSP = 'rtsp',
    eurlIMAP = 'imap',
    eurlNFS = 'unfs',
    eurlPOP = 'upop',
    eurlLDAP = 'uldp',
    eurlUnknown = 'url?'
};
```

Event Class Constants

Specify the event class for an Apple event.

```
enum {
    kCoreEventClass = 'aevt'
};
```

Constants

`kCoreEventClass`

An Apple event sent by the Mac OS; applications that present a graphical interface to the user should be able to any events sent by the Mac OS that apply to the application.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

Discussion

Apple events are identified by their event class and event ID attributes, each of which specifies an arbitrary four-character code. The event class appears in the `message` field of the event record for an Apple event. For example, certain Apple events that are sent by the Mac OS have the value `'aevt'` in the `message` fields of their event records. This value can be represented with the constant `kCoreEventClass`.

Groups of related Apple events are known as suites. For example, the common events that most applications support are grouped in the Standard Suite. The Standard Suite includes the events of the Core suite (`open application`, `reopen`, `open contents`, `open documents`, `print documents`, and `quit`), as well as such events as `count`, `delete`, and `make`. Suites may use a common event class, but doing so is not required, and does not result in any special treatment by AppleScript or the Apple Event Manager.

AppleScript defines suites that provide terminology for Text, Database, Macintosh Connectivity, and other types of related operations. The terms defined in the AppleScript suite itself make up the largest suite. These terms are global to AppleScript, and are available to your application, even if your `'aete'` resource doesn't explicitly include them.

Event Handler Flags

```
enum {
    kAEDoNotIgnoreHandler = 0x00000000,
    kAEIgnoreAppPhacHandler = 0x00000001,
    kAEIgnoreAppEventHandler = 0x00000002,
    kAEIgnoreSysPhacHandler = 0x00000004,
    kAEIgnoreSysEventHandler = 0x00000008,
    kAEIngoreBuiltInEventHandler = 0x00000010,
    kAEDontDisposeOnResume = 0x80000000
};
```

Event ID Constants

Specify the event ID for an Apple event.

```
enum {
    kAEOpenApplication          = 'oapp',
    kAEReopenApplication        = 'rapp',
    kAEOpenDocuments            = 'odoc',
    kAEPrintDocuments           = 'pdoc',
    kAEOpenContents             = 'ocon',
    kAEQuitApplication          = 'quit',
    kAEAnswer                   = 'ansr',
    kAEApplicationDied          = 'obit',
    kAEShowPreferences          = 'pref'
};
```

Constants

`kAEOpenApplication`

Event that launches an application.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEReopenApplication`

Event that reopens an application. Sent, for example, when your application is running and a user clicks your application icon in the Dock.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEOpenDocuments`

Event that provides an application with a list of documents to open. Sent, for example, when a selects one or more documents for your application in the Finder and double-clicks them.

See also the constant `keyAESearchText` in the enum "keyAEPropData" (page 619).

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEPrintDocuments`

Event that provides an application with a list of documents to print.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEOpenContents`

Event that provides an application with dragged content, such as text or an image. Sent, for example, when a user drags an image file onto your application's icon in the Dock. The application can use the content as desired—for example, if no document is currently open, it might open a new document and insert the provided text or image.

For more information, see "Handling Apple Events Sent by the Mac OS" in "Responding to Apple Events" in Apple Events Programming Guide.

Available in Mac OS X v10.4 and later.

Declared in `AppleEvents.h`.

`kAEQuitApplication`

Event that causes the application to quit.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEAnswer`
: Event that is a reply Apple event.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAEApplicationDied`
: Event sent by the Process Manager to an application that launched another application when the launched application quits or terminates.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAEShowPreferences`
: Event sent by the Mac OS X to a process when the user chooses the Preferences item for that process.

  Carbon applications that handle the Preferences command can install an Apple event handler for this event, but they more commonly install a Carbon event handler for `kEventCommandProcess` and check for the `kHICommandPreferences` command ID.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

Discussion
Apple events are identified by their event class and event ID attributes. The event ID is the attribute that identifies the particular Apple event within its event class. In conjunction with the event class, the event ID uniquely identifies the Apple event and communicates what action the Apple event should perform. The event ID appears in the `where` field of the event record for an Apple event. For example, an event with ID `kAEOpenApplication` and class `kCoreEventClass` is an event sent by the Mac OS that launches an application.

Only a small number of event IDs are shown here. For a more complete listing, see the Apple Event Manager and Open Scripting Architecture header files.

Event Source Constants

Identify how an Apple event was delivered.

```
enum {
    kAEUnknownSource = 0,
    kAEDirectCall = 1,
    kAESameProcess = 2,
    kAELocalProcess = 3,
    kAERemoteProcess = 4
};
```

Constants
`kAEUnknownSource`
: The source of the Apple event is unknown.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAEDirectCall`
: The source of the Apple event is a direct call that bypassed the PPC Toolbox.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAESameProcess`
: The source of the Apple event is the same application that received the event (the target application and the source application are the same).

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAELocalProcess`
: The source application is another process on the same computer as the target application.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAERemoteProcess`
: The source application is a process on a remote computer on the network.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

Discussion
For an example of how you might use these constants with the `AEGetAttributePtr` (page 430) function, see the data type `AEEventSource` (page 554).

Declared In
`AppleEvents.h`

Factoring Constants

```
enum {
    kAEDescListFactorNone = 0,
    kAEDescListFactorType = 4,
    kAEDescListFactorTypeAndSize = 8
};
```

Discussion
These constants have no effect in Mac OS X v10.2 and later.

ID Constants for the AECreateAppleEvent Function

Specify values for the ID parameters of the `AECreateAppleEvent` function.

```
enum {
    kAutoGenerateReturnID = -1,
    kAnyTransactionID = 0
};
```

Constants
`kAutoGenerateReturnID`
: If you pass this value for the `returnID` parameter of the `AECreateAppleEvent` (page 416) function, the Apple Event Manager assigns to the created Apple event a return ID that is unique to the current session.

  Available in Mac OS X v10.0 and later.

  Declared in `AEDataModel.h`.

`kAnyTransactionID`

You pass this value for the `transactionID` parameter of the `AECreateAppleEvent` (page 416) function if the Apple event is not one of a series of interdependent Apple events.

A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
You use these constants with the `AECreateAppleEvent` (page 416) function.

Key Form and Descriptor Type Object Specifier Constants

Specify possible values for the `keyAEKeyForm` field of an object specifier, as well as descriptor types used in resolving object specifiers.

```
enum {
    formAbsolutePosition = 'indx',
    formRelativePosition = 'rele',
    formTest = 'test',
    formRange = 'rang',
    formPropertyID = 'prop',
    formName = 'name',
    typeObjectSpecifier = 'obj ',
    typeObjectBeingExamined = 'exmn',
    typeCurrentContainer = 'ccnt',
    typeToken = 'toke',
    typeRelativeDescriptor = 'rel ',
    typeAbsoluteOrdinal = 'abso',
    typeIndexDescriptor = 'inde',
    typeRangeDescriptor = 'rang',
    typeLogicalDescriptor = 'logi',
    typeCompDescriptor = 'cmpd',
    typeOSLTokenList = 'ostl'
};
```

Constants
`formAbsolutePosition`

An integer or other constant indicating the position of one or more elements in relation to the beginning or end of their container. The key data consists of an integer that specifies either an offset or an ordinal position.

For descriptor type `typeAbsoluteOrdinal`, the data consists of one of the constants `kAEFirst`, `kAEMiddle`, `kAELast`, `kAEAny`, or `kAEAll`, which are described in `AEDisposeToken` (page 425).

For other descriptor types, the data can be coerced to either a positive integer, indicating the offset of the requested element from the beginning of the container, or a negative integer, indicating its offset from the end of the container.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formRelativePosition`

Specifies an element position either immediately before or immediately after a container, not inside it. The key data is specified by a descriptor of type `typeEnumerated` whose data consists of one of the constants `kAENext` and `kAEPrevious`, which are described in `AEDisposeToken` (page 425).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formTest`

Specifies a test. The key data is specified by either a comparison descriptor or a logical descriptor.

The Apple Event Manager internally translates object specifiers of key form `formTest` into object specifiers of key form `formWhose` to optimize resolution of object specifiers. This involves collapsing the key form and key data from two object specifiers in a container hierarchy into one object specifier with the key form `formWhose`.

See also `AEDisposeToken` (page 425), "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 575), `CreateCompDescriptor` (page 498), and `CreateLogicalDescriptor` (page 499).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formRange`

Specifies a group of elements between two other elements. The key data is specified by a range descriptor, which is a coerced Apple event record of type `typeRangeDescriptor` that identifies two Apple event objects marking the beginning and end of a range of elements.

The data for a range descriptor consists of two keyword-specified descriptors with the keywords `keyAERangeStart` and `keyAERangeStop`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formPropertyID`

Specifies the property ID for an element's property.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formName`

Specifies the Apple event object by name.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeObjectSpecifier`

Specifies a descriptor used with the `keyAEContainer` keyword in a keyword-specified descriptor. The key data for the descriptor is an object specifier.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeObjectBeingExamined`

Specifies a descriptor that acts as a placeholder for each of the successive elements in a container when the Apple Event Manager tests those elements one at a time. The descriptor has a null data storage pointer. This descriptor type is used only with `formTest`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeCurrentContainer`
Specifies a container for an element that demarcates one boundary in a range. The descriptor has a null data storage pointer. This descriptor type is used only with `formRange`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeToken`
Specifies a descriptor whose data storage pointer refers to a structure of type `AEDisposeToken` (page 425).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeRelativeDescriptor`
Specifies a descriptor whose data consists of one of the constants `kAENext` or `kAEPrevious`, which are described in `AEDisposeToken` (page 425). Used with `formRelativePosition`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeAbsoluteOrdinal`
Specifies a descriptor whose data consists of one of the constants `kAEFirst`, `kAEMiddle`, `kAELast`, `kAEAny`, or `kAEAll`, which are described in `AEDisposeToken` (page 425). Used with `formAbsolutePosition`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeIndexDescriptor`
Specifies a descriptor whose data indicates an indexed position within a range of values.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeRangeDescriptor`
Specifies a range descriptor that identifies two Apple event objects marking the beginning and end of a range of elements. The data for a range descriptor consists of two keyword-specified descriptors with the keywords `keyAERangeStart` and `keyAERangeStop`, respectively, which specify the first Apple event object in the desired range and the last Apple event object in the desired range.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeLogicalDescriptor`
Specifies a logical descriptor. Data is one of the constants described in `AEDisposeToken` (page 425).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeCompDescriptor`
Specifies a comparison descriptor. Data is one of the constants described in `AEDisposeToken` (page 425).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeOSLTokenList`
Specifies a descriptor whose data consists of a list of tokens. (Token is defined in `AEDisposeToken` (page 425).)

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Discussion
The constants in this enum that begin with "form" specify the key form for an object specifier. The key form indicates how key data should be interpreted. Key form is one of the keyword-specified descriptors described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 575).

The constants in this enum that begin with "type" specify descriptor types used in resolving object specifiers. An object specifier is a coerced Apple event record of descriptor type `typeObjectSpecifier` whose data consists of the four keyword-specified descriptors described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 575). One of those four keyword-specified descriptors has the type `keyAEKeyData`. This descriptor can contain data or nested descriptors specified by any of the descriptor type constants defined here (or by types defined by your application).

Keyword Attribute Constants

Specify keyword values for Apple event attributes.

```
enum {
    keyTransactionIDAttr = 'tran',
    keyReturnIDAttr = 'rtid',
    keyEventClassAttr = 'evcl',
    keyEventIDAttr = 'evid',
    keyAddressAttr = 'addr',
    keyOptionalKeywordAttr = 'optk',
    keyTimeoutAttr = 'timo',
    keyInteractLevelAttr = 'inte',
    keyEventSourceAttr = 'esrc',
    keyMissedKeywordAttr = 'miss',
    keyOriginalAddressAttr = 'from',
    keyAcceptTimeoutAttr = 'actm',
    keyReplyRequestedAttr = 'repq'
};
```

Constants
`keyTransactionIDAttr`
Transaction ID identifying a series of Apple events that are part of one transaction.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyReturnIDAttr`
Return ID for a reply Apple event.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyEventClassAttr`
Event class of an Apple event. See `AEAddressDesc` (page 551).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyEventIDAttr`
Event ID of an Apple event. See `AEAddressDesc` (page 551).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyAddressAttr`
Address of a target or client application. See also `AEAddressDesc` (page 551).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyOptionalKeywordAttr`
List of keywords for parameters of an Apple event that should be treated as optional by the target application.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyTimeoutAttr`
Length of time, in ticks, that the client will wait for a reply or a result from the server.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyInteractLevelAttr`
Settings for when to allow the Apple Event Manager to bring a server application to the foreground, if necessary, to interact with the user. See `AEAddressDesc` (page 551). (Read only.)

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyEventSourceAttr`
Nature of the source application. (Read only.)

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyMissedKeywordAttr`
Keyword for first required parameter remaining in an Apple event. (Read only.)

After extracting all known Apple event parameters from an event, your handler should check whether the `keyMissedKeywordAttr` attribute exists. If so, your handler has not retrieved all the parameters that the source application considered to be required, and it should return an error.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyOriginalAddressAttr`
Address of original source of Apple event if the event has been forwarded (available only in version 1.01 or later versions of the Apple Event Manager). See also `AEAddressDesc` (page 551).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyReplyRequestedAttr`
A Boolean value indicating whether the Apple event expects to be replied to.

Available in Mac OS X v10.3 and later.

Declared in `AEDataModel.h`.

Discussion
These constants are keyword constants for Apple event attributes. An Apple event consists of attributes (which identify the Apple event and denote its task) and, often, parameters (which contain information to be used by the target application). An Apple event attribute is a descriptor that identifies the event class, event ID, target application, or some other characteristic of the Apple event. Taken together, the attributes of an Apple event denote the task to be performed on any data specified in the Apple event's parameters.

Keywords are arbitrary names used by the Apple Event Manager to keep track of various descriptors. Your application cannot examine the contents of an Apple event directly. Instead, you call Apple Event Manager routines such as those described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 402) to request attributes and parameters by keyword.

See also "Keyword Parameter Constants" (page 595).

Version Notes
The constant `keyReplyRequestedAttr` was added in Mac OS X version 10.3.

Keyword Parameter Constants

Specify keyword values for Apple event parameters, as well as information for the `AEManagerInfo` function to retrieve. Some common key word values are shown here.

```
enum {
    keyDirectObject = '----',
    keyErrorNumber = 'errn',
    keyErrorString = 'errs',
    keyProcessSerialNumber = 'psn ',
    keyPreDispatch = 'phac',
    keySelectProc = 'selh',
    keyAERecorderCount = 'recr',
    keyAEVersion = 'vers'
};
```

Constants

`keyDirectObject`
Direct parameter. Usually specifies the data to be acted upon by the target application.
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyErrorNumber`
Error number. Often used to extract error information from a reply Apple event.
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyErrorString`
Error string. Often used to extract error information from a reply Apple event to display to the user.
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyProcessSerialNumber`
Process serial number. See also `AEManagerInfo` (page 454).
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyPreDispatch`
: A predispatch handler (an Apple event handler that the Apple Event Manager calls immediately before it dispatches an Apple event). See also "Managing Special Handler Dispatch Tables" (page 404).

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`keySelectProc`
: You pass this value in the `functionClass` parameter of the `AEManagerInfo` (page 454) function to disable the Object Support Library. Disabling the Object Support Library is not recommended.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`keyAERecorderCount`
: Used with the `keyword` parameter of the `AEManagerInfo` (page 454) function. If you pass this value, on return, the `result` parameter supplies the number of processes that are currently recording Apple events.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`keyAEVersion`
: Used with the `keyword` parameter of the `AEManagerInfo` (page 454) function. If you pass this value, on return, the `result` parameter supplies version information for the Apple Event Manager, in NumVersion format.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

Discussion
These constants are keyword constants for Apple event parameters. An Apple event consists of attributes (which identify the Apple event and denote its task) and, often, parameters (which contain information to be used by the target application). Taken together, the attributes of an Apple event denote the task to be performed on any data specified in the Apple event's parameters.

Keywords are arbitrary names used by the Apple Event Manager to keep track of various descriptors. Your application cannot examine the contents of an Apple event directly. Instead, you call Apple Event Manager routines such as those described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 402) to request attributes and parameters by keyword.

See also "Keyword Attribute Constants" (page 593).

Launch Apple Event Constants

In a `kAEOpenApplication` event, specify information about how the receiving application was launched.

```
enum {
    keyAELaunchedAsLogInItem = 'lgit',
    keyAELaunchedAsServiceItem = 'svit'
};
```

Constants

`keyAELaunchedAsLogInItem`

If present in a `kAEOpenApplication` event, the receiving application was launched as a login item and should only perform actions suitable to that environment—for example, it probably shouldn't open an untitled document.

Available in Mac OS X v10.5 and later.

Declared in `AERegistry.h`.

`keyAELaunchedAsServiceItem`

If present in a `kAEOpenApplication` event, the receiving application was launched as a service item and should only perform actions suitable to that environment—for example, it probably shouldn't open an untitled document.

Available in Mac OS X v10.5 and later.

Declared in `AERegistry.h`.

Special Considerations

Although these constants were not publicly defined in Mac OS X version 10.4, corresponding information was provided in `kAEOpenApplication` Apple events sent by that version of the OS. Therefore your application, running on Mac OS X version 10.4 or later, can examine the open application Apple event to determine if the application was launched as a login item or a service item. However, for version 10.4, you will have to define these constants in your own code file.

You check for a `keyAEPropData` parameter of the `kAEOpenApplication` Apple event, with a data value that matches `keyAELaunchedAsLogInItem` or `keyAELaunchedAsServiceItem`.

Declared In

`AERegistry.h`

Numeric Descriptor Type Constants

Specify types for numeric descriptors.

```
enum {
    typeSInt16 = 'shor',
    typeUInt16 = 'ushr',
    typeSInt32 = 'long',
    typeUInt32 = 'magn',
    typeSInt64 = 'comp',
    typeUInt64 = 'ucom',
    typeIEEE32BitFloatingPoint = 'sing',
    typeIEEE64BitFloatingPoint = 'doub',
    type128BitFloatingPoint = 'ldbl',
    typeDecimalStruct = 'decm'
};
```

Constants

`typeSInt16`
16-bit signed integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeUInt16`
16-bit unsigned integer.
Available in Mac OS X v10.5 and later.
Declared in `AEDataModel.h`.

`typeSInt32`
32-bit signed integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeUInt32`
32-bit unsigned integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeSInt64`
64-bit signed integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeUInt64`
64-bit unsigned integer.
Available in Mac OS X v10.5 and later.
Declared in `AEDataModel.h`.

`typeIEEE32BitFloatingPoint`
32-bit floating point value.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeIEEE64BitFloatingPoint`
64-bit floating point value.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`type128BitFloatingPoint`
128-bit floating point value.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeDecimalStruct`
Decimal.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
The constants described here specify the data type for a descriptor and show the kind of numeric data stored in a descriptor with that type. These constants are preferred over their older equivalents described in "typeSMInt" (page 632).

Descriptors are the building blocks used by the Apple Event Manager to construct Apple event attributes and parameters. A descriptor is a data structure of type `AEDesc` (page 546), which consists of data storage and a descriptor type that identifies the type of the data. A descriptor type is defined by the data type `DescType` (page 560).

AppleScript defines descriptor type constants for a wide variety of common data types. For additional types, see "Descriptor Type Constants" (page 581) and "Other Descriptor Type Constants" (page 601). For a complete listing, including data types such as units of length, weight, and volume, see the Apple Event Manager and Open Scripting Architecture header files.

Declared In
`AEDataModel.h`

Object Class ID Constants

Specify the object class for an Apple event object.

```
enum {
    cParagraph = 'cpar',
    cPICT = 'PICT',
    cPixel = 'cpxl',
    cPixelMap = 'cpix',
    cPolygon = 'cpgn',
    cProperty = 'prop',
    cQDPoint = 'QDpt',
    cQDRectangle = 'qdrt',
    cRectangle = 'crec',
    cRGBColor = 'cRGB',
    cRotation = 'trot',
    cRoundedRectangle = 'crrc',
    cRow = 'crow',
    cSelection = 'csel',
    cShortInteger = 'shor',
    cTable = 'ctbl',
    cText = 'ctxt',
    cTextFlow = 'cflo',
    cTextStyles = 'tsty',
    cType = 'type'
};
```

Constants

`cParagraph`

A paragraph of text.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cPICT`

A PICT format figure.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cProperty`

A property of any object class.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cRGBColor`

An RGB color value.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

Discussion

The object class of an Apple event object is identified by an object class ID. For example, the object class for an object specifier that specifies an RGB color value is the four-character code `'cRGB'`, which can be represented by the constant `cRGBColor`.

AppleScript defines constants for a wide variety of common object classes, though only a small number are shown here. For a more complete listing, see the Apple Event Manager and Open Scripting Architecture header files.

Other Descriptor Type Constants

Specify types for Boolean and character descriptors.

```
enum {
    typeBoolean = 'bool',
    typeChar = 'TEXT'
};
```

Constants

`typeBoolean`

Boolean value—single byte with value 0 or 1.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeChar`

Unterminated string of system script characters.

See the Version Notes section below for important information.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

The constants described here specify the data type for a descriptor and show the kind of data stored in a descriptor with that type.

Descriptors are the building blocks used by the Apple Event Manager to construct Apple event attributes and parameters. A descriptor is a data structure of type `AEDesc` (page 546), which consists of data storage and a descriptor type that identifies the type of the data. A descriptor type is defined by the data type `DescType` (page 560).

AppleScript defines descriptor type constants for a wide variety of common data types. For additional types, see "Descriptor Type Constants" (page 581) and "Numeric Descriptor Type Constants" (page 597). For a complete listing, including data types such as units of length, weight, and volume, see the Apple Event Manager and Open Scripting Architecture header files.

Version Notes

On Mac OS X `typeChar` type is deprecated in favor of `typeUTF8Text` or `typeUTF16ExternalRepresentation`. For more information, see `typeUTF16ExternalRepresentation` (page 635).

Priority Constants for the AESend Function (Deprecated in Mac OS X)

Specify a value for the `sendPriority` parameter of the `AESend` function. (**Deprecated.** Not used in Mac OS X.)

```
enum {
    kAENormalPriority = 0x00000000,
    kAEHighPriority = 0x00000001
};
```

Constants

`kAENormalPriority`

The Apple Event Manager posts the event at the end of the event queue of the server process and the server processes the Apple event as soon as it has the opportunity.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEHighPriority`

The Apple Event Manager posts the event at the beginning of the event queue of the server process.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

For related information, see the `AESend` (page 476) function and "AESendMode" (page 566).

Version Notes

The `sendPriority` parameter of the `AESend` function is deprecated in Mac OS X.

Remote Process Dictionary Keys

Used to extract information from dictionaries with entries that describe remote processes.

```
extern const CFStringRef kAERemoteProcessURLKey;
extern const CFStringRef kAERemoteProcessNameKey;
extern const CFStringRef kAERemoteProcessUserIDKey;
extern const CFStringRef kAERemoteProcessProcessIDKey;
```

Constants

`kAERemoteProcessURLKey`

Use this key to obtain the full URL to the remote process, as a `CFURLRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

`kAERemoteProcessNameKey`

Use this key to obtain the visible name of the remote process, in the localization supplied by the server, as a `CFStringRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

`kAERemoteProcessUserIDKey`

Use this key to obtain the user ID of the remote process, if available; if so, returned as a `CFNumberRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

`kAERemoteProcessProcessIDKey`
Use this key to obtain the process ID of the remote process, if available; if so, returned as a `CFNumberRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

Declared In
`AppleEvents.h`

Resume Event Dispatch Constants

Specify event dispatching information to the `AEResumeTheCurrentEvent` function.

```
enum {
    kAENoDispatch = 0,
    kAEUseStandardDispatch = 0xFFFFFFFF
};
```

Constants
`kAENoDispatch`
Tells the Apple Event Manager that the Apple event has been completely processed and need not be dispatched.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

`kAEUseStandardDispatch`
Tells the Apple Event Manager to dispatch the resumed event using the standard dispatching scheme it uses for other Apple events.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

Discussion
You call the `AEResumeTheCurrentEvent` (page 474) function to inform the Apple Event Manager that your application wants to resume the handling of a previously suspended Apple event or that it has completed the handling of the Apple event. You pass one of the constants described here in the `dispatcher` parameter to provide dispatching information to the Apple Event Manager. You can also pass a handler universal procedure pointer.

Special Handler Callback Constants

Specify an object callback function to install, get, or remove from the special handler dispatch table.

```
enum {
    keyAERangeStart = 'star',
    keyAERangeStop = 'stop',
    keyDisposeTokenProc = 'xtok',
    keyAECompareProc = 'cmpr',
    keyAECountProc = 'cont',
    keyAEMarkTokenProc = 'mkid',
    keyAEMarkProc = 'mark',
    keyAEAdjustMarksProc = 'adjm',
    keyAEGetErrDescProc = 'indc'
};
```

Constants

`keyAERangeStart`
Specifies the first Apple event object in a desired range.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAERangeStop`
Specifies the last Apple event object in the desired range.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyDisposeTokenProc`
Token disposal function. See `OSLDisposeTokenProcPtr` (page 539).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAECompareProc`
Object-comparison function. See `OSLCompareProcPtr` (page 536).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAECountProc`
Object-counting function. See `OSLCountProcPtr` (page 538).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAEMarkTokenProc`
Mark token function. See `OSLGetMarkTokenProcPtr` (page 542).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAEMarkProc`
Object-marking function. See `OSLMarkProcPtr` (page 544).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAEAdjustMarksProc`
Mark-adjusting function. See `OSLAdjustMarksProcPtr` (page 535).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAEGetErrDescProc`
: Get error descriptor callback function. See `OSLGetErrDescProcPtr` (page 541).

  Available in Mac OS X v10.0 and later.

  Declared in `AEObjects.h`.

Discussion
You use these constants with the `AEInstallSpecialHandler` (page 452), `AEGetSpecialHandler` (page 446), or `AERemoveSpecialHandler` (page 471) functions.

Timeout Constants

Specify a timeout value.

```
enum {
    kAEDefaultTimeout = -1,
    kNoTimeOut = -2
};
```

Constants
`kAEDefaultTimeout`
: The timeout value is determined by the Apple Event Manager. The default timeout value is about one minute.

  Available in Mac OS X v10.0 and later.

  Declared in `AEDataModel.h`.

`kNoTimeOut`
: Your application is willing to wait indefinitely. Most commonly, you instead provide a timeout value (in ticks) that will provide a reasonable amount of time for the current operation.

  Available in Mac OS X v10.0 and later.

  Declared in `AEDataModel.h`.

Discussion
Your application can use these constants when it calls the `AEInteractWithUser` (page 453) function, or it can supply the specific amount of time (in ticks) that your handler is willing to wait for a response from the user. You can also use the constants with the `AESend` (page 476) function.

User Interaction Level Constants

Specify to the `AESetInteractionAllowed` function the conditions under which your application is willing to interact with the user.

```
enum {
    kAEInteractWithSelf = 0,
    kAEInteractWithLocal = 1,
    kAEInteractWithAll = 2
};
```

Constants

`kAEInteractWithSelf`

Indicates that the server application may interact with the user in response to an Apple event only when the client application and server application are the same—that is, only when your application is sending the Apple event to itself.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

`kAEInteractWithLocal`

Indicates that your server application may interact with the user in response to an Apple event only if the client application is on the same computer as the server application. This is the default value if your application has not called the `AESetInteractionAllowed` (page 479) function to set the interaction level explicitly.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

`kAEInteractWithAll`

Indicates that your server application may interact with the user in response to an Apple event sent from any client application on any computer.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

Discussion

If your application does not set the user interaction level by calling the `AESetInteractionAllowed` (page 479) function, the Apple Event Manager uses `kAEInteractWithLocal` as the default value.

Declared In

`AERegistry.h`

Whose Test Constants

```
enum {
    typeWhoseDescriptor = 'whos',
    formWhose = 'whos',
    typeWhoseRange = 'wrng',
    keyAEWhoseRangeStart = 'wstr',
    keyAEWhoseRangeStop = 'wstp',
    keyAEIndex = 'kidx',
    keyAETest = 'ktst'
};
```

Constants

`formWhose`

Specifies a container of one or more objects and a test to perform on the objects.

The key data for `formWhose` is specified by a whose descriptor, which is a coerced Apple event record of descriptor type `typeWhoseDescriptor`. The data for a whose descriptor consists of two keyword-specified descriptors with the keywords `keyAEIndex` and `keyAETest`.

See also the description for `formTest`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

kAEDoObjectsExist

```
enum {
    kAEDoObjectsExist = 'doex',
    kAEDoScript = 'dosc',
    kAEDrag = 'drag',
    kAEDuplicateSelection = 'sdup',
    kAEEditGraphic = 'edit',
    kAEEmptyTrash = 'empt',
    kAEEnd = 'end ',
    kAEEndsWith = 'ends',
    kAEEndTransaction = 'endt',
    kAEEquals = '=   ',
    kAEExpanded = 'pexp',
    kAEFast = 'fast',
    kAEFinderEvents = 'FNDR',
    kAEFormulaProtect = 'fpro',
    kAEFullyJustified = 'full',
    kAEGetClassInfo = 'qobj',
    kAEGetData = 'getd',
    kAEGetDataSize = 'dsiz',
    kAEGetEventInfo = 'gtei',
    kAEGetInfoSelection = 'sinf'
};
```

Constants

`kAEEndsWith`

The value of `operand1` ends with the value of `operand2` (for example, the string `"operand"` ends with the string `"and"`).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEEquals`
The value of `operand1` is equal to the value of `operand2`

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEFinderEvents`
An event that the Finder accepts.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

kAEDebugPOSTHeader

```
enum {
    kAEDebugPOSTHeader = 0x01,
    kAEDebugReplyHeader = 0x02,
    kAEDebugXMLRequest = 0x04,
    kAEDebugXMLResponse = 0x08,
    kAEDebugXMLDebugAll = 0xFFFFFFFF
};
```

kAEGetPrivilegeSelection

```
enum {
    kAEGetPrivilegeSelection = 'sprv',
    kAEGetSuiteInfo = 'gtsi',
    kAEGreaterThan = '>   ',
    kAEGreaterThanEquals = '>=  ',
    kAEGrow = 'grow',
    kAEHidden = 'hidn',
    kAEHiQuality = 'hiqu',
    kAEImageGraphic = 'imgr',
    kAEIsUniform = 'isun',
    kAEItalic = 'ital',
    kAELeftJustified = 'left',
    kAELessThan = '<   ',
    kAELessThanEquals = '<=  ',
    kAELowercase = 'lowc',
    kAEMakeObjectsVisible = 'mvis',
    kAEMiscStandards = 'misc',
    kAEModifiable = 'modf',
    kAEMove = 'move',
    kAENo = 'no  ',
    kAENoArrow = 'arno'
};
```

Constants

`kAEGreaterThan`
The value of `operand1` is greater than the value of `operand2`.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEGreaterThanEquals`
The value of `operand1` is greater than or equal to the value of `operand2`.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAELessThanEquals`
The value of `operand1` is less than or equal to the value of `operand2`.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

kAEHandleArray

```
enum {
    kAEHandleArray = 2
};
```

Constants

`kAEHandleArray`
Array items consist of handles to data of the same type and possibly variable size.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

kAEInfo

```
enum {
    kAEInfo = 11,
    kAEMain = 0,
    kAESharing = 13
};
```

kAEInternetSuite

```
enum {
    kAEInternetSuite = 'gurl',
    kAEISWebStarSuite = 'WWW'
};
```

kAEISGetURL

```
enum {
    kAEISGetURL = 'gurl',
    KAEISHandleCGI = 'sdoc'
};
```

kAEISHTTPSearchArgs

```
enum {
    kAEISHTTPSearchArgs = 'kfor',
    kAEISPostArgs = 'post',
    kAEISMethod = 'meth',
    kAEISClientAddress = 'addr',
    kAEISUserName = 'user',
    kAEISPassword = 'pass',
    kAEISFromUser = 'frmu',
    kAEISServerName = 'svnm',
    kAEISServerPort = 'svpt',
    kAEISScriptName = 'scnm',
    kAEISContentType = 'ctyp',
    kAEISReferrer = 'refr',
    kAEISUserAgent = 'Agnt',
    kAEISAction = 'Kact',
    kAEISActionPath = 'Kapt',
    kAEISClientIP = 'Kcip',
    kAEISFullRequest = 'Kfrq'
};
```

kAELogOut

```
enum {
    kAELogOut = 'logo',
    kAEReallyLogOut = 'rlgo',
    kAEShowRestartDialog = 'rrst',
    kAEShowShutdownDialog = 'rsdn'
```

```
};
```

kAEMenuClass

```
enum {
    kAEMenuClass = 'menu',
    kAEMenuSelect = 'mhit',
    kAEMouseDown = 'mdwn',
    kAEMouseDownInBack = 'mdbk',
    kAEKeyDown = 'kdwn',
    kAEResized = 'rsiz',
    kAEPromise = 'prom'
};
```

kAEMouseClass

```
enum {
    kAEMouseClass = 'mous',
    kAEDown = 'down',
    kAEUp = 'up  ',
    kAEMoved = 'move',
    kAEStoppedMoving = 'stop',
    kAEWindowClass = 'wind',
    kAEUpdate = 'updt',
    kAEActivate = 'actv',
    kAEDeactivate = 'dact',
    kAECommandClass = 'cmnd',
    kAEKeyClass = 'keyc',
    kAERawKey = 'rkey',
    kAEVirtualKey = 'keyc',
    kAENavigationKey = 'nave',
    kAEAutoDown = 'auto',
    kAEApplicationClass = 'appl',
    kAESuspend = 'susp',
    kAEResume = 'rsme',
    kAEDiskEvent = 'disk',
    kAENullEvent = 'null',
    kAEWakeUpEvent = 'wake',
    kAEScrapEvent = 'scrp',
    kAEHighLevel = 'high'
};
```

kAENonmodifiable

```
enum {
    kAENonmodifiable = 'nmod',
    kAEOpen = 'odoc',
    kAEOpenSelection = 'sope',
    kAEOutline = 'outl',
    kAEPageSetup = 'pgsu',
    kAEPaste = 'past',
    kAEPlain = 'plan',
    kAEPrint = 'pdoc',
```

```
    kAEPrintSelection = 'spri',
    kAEPrintWindow = 'pwin',
    kAEPutAwaySelection = 'sput',
    kAEQDAddOver = 'addo',
    kAEQDAddPin = 'addp',
    kAEQDAdMax = 'admx',
    kAEQDAdMin = 'admn',
    kAEQDBic = 'bic ',
    kAEQDBlend = 'blnd',
    kAEQDCopy = 'cpy ',
    kAEQDNotBic = 'nbic',
    kAEQDNotCopy = 'ncpy'
};
```

kAEQDNotOr

```
enum {
    kAEQDNotOr = 'ntor',
    kAEQDNotXor = 'nxor',
    kAEQDOr = 'or  ',
    kAEQDSubOver = 'subo',
    kAEQDSubPin = 'subp',
    kAEQDSupplementalSuite = 'qdsp',
    kAEQDXor = 'xor ',
    kAEQuickdrawSuite = 'qdrw',
    kAEQuitAll = 'quia',
    kAERedo = 'redo',
    kAERegular = 'regl',
    kAEReplace = 'rplc',
    kAERequiredSuite = 'reqd',
    kAERestart = 'rest',
    kAERevealSelection = 'srev',
    kAERevert = 'rvrt',
    kAERightJustified = 'rght',
    kAESave = 'save',
    kAESelect = 'slct',
    kAESetData = 'setd'
};
```

kAESetPosition

```
enum {
    kAESetPosition = 'posn',
    kAEShadow = 'shad',
    kAEShowClipboard = 'shcl',
    kAEShutDown = 'shut',
    kAESleep = 'slep',
    kAESmallCaps = 'smcp',
    kAESpecialClassProperties = 'c@#!',
    kAEStrikethrough = 'strk',
    kAESubscript = 'sbsc',
    kAESuperscript = 'spsc',
    kAETableSuite = 'tbls',
    kAETextSuite = 'TEXT',
    kAETransactionTerminated = 'ttrm',
```

```
    kAEUnderline = 'undl',
    kAEUndo = 'undo',
    kAEWholeWordEquals = 'wweq',
    kAEYes = 'yes ',
    kAEZoom = 'zoom'
};
```

kAESocks4Protocol

```
enum {
    kAESocks4Protocol = 4,
    kAESocks5Protocol = 5
};
```

kAEUseHTTPProxyAttr

Web Services Proxy support—these constants should be added as attributes of the event that is being sent (not as part of the direct object).

```
enum {
    kAEUseHTTPProxyAttr = 'xupr',
    kAEHTTPProxyPortAttr = 'xhtp',
    kAEHTTPProxyHostAttr = 'xhth'
};
```

Constants

`kAEUseHTTPProxyAttr`

A value of type `typeBoolean`. Specifies whether to manually specify the proxy host and port. Defaults to `true`.

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

`kAEHTTPProxyPortAttr`

A value of type `typeSInt32`.

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

`kAEHTTPProxyHostAttr`

A value of type `typeChar` or `typeUTF8Text`.

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

kAEUserTerminology

```
enum {
    kAEUserTerminology = 'aeut',
    kAETerminologyExtension = 'aete',
    kAEScriptingSizeResource = 'scsz',
    kAEOSAXSizeResource = 'osiz'
};
```

kAEUseSocksAttr

```
enum {
    kAEUseSocksAttr = 'xscs',
    kAESocksProxyAttr = 'xsok',
    kAESocksHostAttr = 'xshs',
    kAESocksPortAttr = 'xshp',
    kAESocksUserAttr = 'xshu',
    kAESocksPasswordAttr = 'xshw'
};
```

kAEUTHasReturningParam

```
enum {
    kAEUTHasReturningParam = 31,
    kAEUTOptional = 15,
    kAEUTlistOfItems = 14,
    kAEUTEnumerated = 13,
    kAEUTReadWrite = 12,
    kAEUTChangesState = 12,
    kAEUTTightBindingFunction = 12,
    kAEUTEnumsAreTypes = 11,
    kAEUTEnumListIsExclusive = 10,
    kAEUTReplyIsReference = 9,
    kAEUTDirectParamIsReference = 9,
    kAEUTParamIsReference = 9,
    kAEUTPropertyIsReference = 9,
    kAEUTNotDirectParamIsTarget = 8,
    kAEUTParamIsTarget = 8,
    kAEUTApostrophe = 3,
    kAEUTFeminine = 2,
    kAEUTMasculine = 1,
    kAEUTPlural = 0
};
```

kAEZoomIn

```
enum {
    kAEZoomIn = 7,
    kAEZoomOut = 8
};
```

kBySmallIcon

```
enum {
    kBySmallIcon = 0,
    kByIconView = 1,
    kByNameView = 2,
    kByDateView = 3,
    kBySizeView = 4,
    kByKindView = 5,
    kByCommentView = 6,
    kByLabelView = 7,
    kByVersionView = 8
};
```

kCaretPosition

```
enum {
    kCaretPosition = 1,
    kRawText = 2,
    kSelectedRawText = 3,
    kConvertedText = 4,
    kSelectedConvertedText = 5,
    kBlockFillText = 6,
    kOutlineText = 7,
    kSelectedText = 8
};
```

Version Notes
Starting in Mac OS X v10.4, use the constants defined in "kTSMHiliteCaretPosition" (page 624) in place of these constants.

kConnSuite

```
enum {
    kConnSuite = 'macc',
    cDevSpec = 'cdev',
    cAddressSpec = 'cadr',
    cADBAddress = 'cadb',
    cAppleTalkAddress = 'cat ',
    cBusAddress = 'cbus',
    cEthernetAddress = 'cen ',
    cFireWireAddress = 'cfw ',
    cIPAddress = 'cip ',
    cLocalTalkAddress = 'clt ',
    cSCSIAddress = 'cscs',
    cTokenRingAddress = 'ctok',
    cUSBAddress = 'cusb',
    pDeviceType = 'pdvt',
    pDeviceAddress = 'pdva',
    pConduit = 'pcon',
    pProtocol = 'pprt',
    pATMachine = 'patm',
    pATZone = 'patz',
    pATType = 'patt',
    pDottedDecimal = 'pipd',
    pDNS = 'pdns',
    pPort = 'ppor',
    pNetwork = 'pnet',
    pNode = 'pnod',
    pSocket = 'psoc',
    pSCSIBus = 'pscb',
    pSCSILUN = 'pslu',
    eDeviceType = 'edvt',
    eAddressSpec = 'eads',
    eConduit = 'econ',
    eProtocol = 'epro',
    eADB = 'eadb',
    eAnalogAudio = 'epau',
    eAppleTalk = 'epat',
    eAudioLineIn = 'ecai',
    eAudioLineOut = 'ecal',
    eAudioOut = 'ecao',
    eBus = 'ebus',
    eCDROM = 'ecd ',
    eCommSlot = 'eccm',
    eDigitalAudio = 'epda',
    eDisplay = 'edds',
    eDVD = 'edvd',
    eEthernet = 'ecen',
    eFireWire = 'ecfw',
    eFloppy = 'efd ',
    eHD = 'ehd ',
    eInfrared = 'ecir',
    eIP = 'epip',
    eIrDA = 'epir',
    eIRTalk = 'epit',
    eKeyboard = 'ekbd',
    eLCD = 'edlc',
    eLocalTalk = 'eclt',
```

```
    eMacIP = 'epmi',
    eMacVideo = 'epmv',
    eMicrophone = 'ecmi',
    eModemPort = 'ecmp',
    eModemPrinterPort = 'empp',
    eModem = 'edmm',
    eMonitorOut = 'ecmn',
    eMouse = 'emou',
    eNuBusCard = 'ednb',
    eNuBus = 'enub',
    ePCcard = 'ecpc',
    ePCIbus = 'ecpi',
    ePCIcard = 'edpi',
    ePDSslot = 'ecpd',
    ePDScard = 'epds',
    ePointingDevice = 'edpd',
    ePostScript = 'epps',
    ePPP = 'eppp',
    ePrinterPort = 'ecpp',
    ePrinter = 'edpr',
    eSvideo = 'epsv',
    eSCSI = 'ecsc',
    eSerial = 'epsr',
    eSpeakers = 'edsp',
    eStorageDevice = 'edst',
    eSVGA = 'epsg',
    eTokenRing = 'etok',
    eTrackball = 'etrk',
    eTrackpad = 'edtp',
    eUSB = 'ecus',
    eVideoIn = 'ecvi',
    eVideoMonitor = 'edvm',
    eVideoOut = 'ecvo'
};
```

keyAEAngle

```
enum {
    keyAEAngle = 'kang',
    keyAEArcAngle = 'parc'
};
```

keyAEBaseAddr

```
enum {
    keyAEBaseAddr = 'badd',
    keyAEBestType = 'pbst',
    keyAEBgndColor = 'kbcl',
    keyAEBgndPattern = 'kbpt',
    keyAEBounds = 'pbnd',
    keyAECellList = 'kclt',
    keyAEClassID = 'clID',
    keyAEColor = 'colr',
    keyAEColorTable = 'cltb',
    keyAECurveHeight = 'kchd',
```

```
    keyAECurveWidth = 'kcwd',
    keyAEDashStyle = 'pdst',
    keyAEData = 'data',
    keyAEDefaultType = 'deft',
    keyAEDefinitionRect = 'pdrt',
    keyAEDescType = 'dstp',
    keyAEDestination = 'dest',
    keyAEDoAntiAlias = 'anta',
    keyAEDoDithered = 'gdit',
    keyAEDoRotate = 'kdrt'
};
```

keyAEDoScale

```
enum {
    keyAEDoScale = 'ksca',
    keyAEDoTranslate = 'ktra',
    keyAEEditionFileLoc = 'eloc',
    keyAEElements = 'elms',
    keyAEEndPoint = 'pend',
    keyAEEventClass = 'evcl',
    keyAEEventID = 'evti',
    keyAEFile = 'kfil',
    keyAEFileType = 'fltp',
    keyAEFillColor = 'flcl',
    keyAEFillPattern = 'flpt',
    keyAEFlipHorizontal = 'kfho',
    keyAEFlipVertical = 'kfvt',
    keyAEFont = 'font',
    keyAEFormula = 'pfor',
    keyAEGraphicObjects = 'gobs',
    keyAEID = 'ID  ',
    keyAEImageQuality = 'gqua',
    keyAEInsertHere = 'insh',
    keyAEKeyForms = 'keyf'
};
```

keyAEHiliteRange

```
enum {
    keyAEHiliteRange = 'hrng',
    keyAEPinRange = 'pnrg',
    keyAEClauseOffsets = 'clau',
    keyAEOffset = 'ofst',
    keyAEPoint = 'gpos',
    keyAELeftSide = 'klef',
    keyAERegionClass = 'rgnc',
    keyAEDragging = 'bool'
};
```

keyAEKeyword

```
enum {
```

```
    keyAEKeyword = 'kywd',
    keyAELevel = 'levl',
    keyAELineArrow = 'arro',
    keyAEName = 'pnam',
    keyAENewElementLoc = 'pnel',
    keyAEObject = 'kobj',
    keyAEObjectClass = 'kocl',
    keyAEOffStyles = 'ofst',
    keyAEOnStyles = 'onst',
    keyAEParameters = 'prms',
    keyAEParamFlags = 'pmfg',
    keyAEPenColor = 'ppcl',
    keyAEPenPattern = 'pppa',
    keyAEPenWidth = 'ppwd',
    keyAEPixelDepth = 'pdpt',
    keyAEPixMapMinus = 'kpmm',
    keyAEPMTable = 'kpmt',
    keyAEPointList = 'ptlt',
    keyAEPointSize = 'ptsz',
    keyAEPosition = 'kpos'
};
```

keyAELeadingEdge

```
enum {
    keyAELeadingEdge = 'klef'
};
```

keyAEPropData

```
enum {
    keyAEPropData = 'prdt',
    keyAEProperties = 'qpro',
    keyAEProperty = 'kprp',
    keyAEPropFlags = 'prfg',
    keyAEPropID = 'prop',
    keyAEProtection = 'ppro',
    keyAERenderAs = 'kren',
    keyAERequestedType = 'rtyp',
    keyAEResult = '----',
    keyAEResultInfo = 'rsin',
    keyAERotation = 'prot',
    keyAERotPoint = 'krtp',
    keyAERowList = 'krls',
    keyAESaveOptions = 'savo',
    keyAEScale = 'pscl',
    keyAEScriptTag = 'psct',
    keyAESearchText = 'stxt',
    keyAEShowWhere = 'show',
    keyAEStartAngle = 'pang',
    keyAEStartPoint = 'pstp',
    keyAEStyles = 'ksty'
};
```

Constants

`keyAESearchText`

Identifies an optional parameter to the `open documents` Apple event, described in "Event ID Constants" (page 586). The parameter contains the search text from the Spotlight search that identified the documents to be opened. The application should make a reasonable effort to display an occurrence of the search text in each opened document—for example by scrolling the text into view.

For more information, see "Handling Apple Events Sent by the Mac OS" in "Responding to Apple Events" in Apple Events Programming Guide.

Available in Mac OS X v10.4 and later.

Declared in `AERegistry.h`.

Version Notes

The constant `keyAESearchText` is available starting in Mac OS X v10.4.

keyAESuiteID

```
enum {
    keyAESuiteID = 'suit',
    keyAEText = 'ktxt',
    keyAETextColor = 'ptxc',
    keyAETextFont = 'ptxf',
    keyAETextPointSize = 'ptps',
    keyAETextStyles = 'txst',
    keyAETextLineHeight = 'ktlh',
    keyAETextLineAscent = 'ktas',
    keyAETheText = 'thtx',
    keyAETransferMode = 'pptm',
    keyAETranslation = 'ptrs',
    keyAETryAsStructGraf = 'toog',
    keyAEUniformStyles = 'ustl',
    keyAEUpdateOn = 'pupd',
    keyAEUserTerm = 'utrm',
    keyAEWindow = 'wndw',
    keyAEWritingCode = 'wrcd'
};
```

keyMenuID

```
enum {
    keyMenuID = 'mid ',
    keyMenuItem = 'mitm',
    keyCloseAllWindows = 'caw ',
    keyOriginalBounds = 'obnd',
    keyNewBounds = 'nbnd',
    keyLocalWhere = 'lwhr'
};
```

keyMiscellaneous

```
enum {
    keyMiscellaneous = 'fmsc',
    keySelection = 'fsel',
    keyWindow = 'kwnd',
    keyWhen = 'when',
    keyWhere = 'wher',
    keyModifiers = 'mods',
    keyKey = 'key ',
    keyKeyCode = 'code',
    keyKeyboard = 'keyb',
    keyDriveNumber = 'drv#',
    keyErrorCode = 'err#',
    keyHighLevelClass = 'hcls',
    keyHighLevelID = 'hid '
};
```

keyReplyPortAttr

```
enum {
    keyReplyPortAttr = 'repp'
};
```

keySOAPStructureMetaData

```
enum {
    keySOAPStructureMetaData = '/smd',
    keySOAPSMDNamespace = 'ssns',
    keySOAPSMDNamespaceURI = 'ssnu',
    keySOAPSMDType = 'sstp'
};
```

keyUserNameAttr

```
enum {
    keyUserNameAttr = 'unam',
    keyUserPasswordAttr = 'pass',
    keyDisableAuthenticationAttr = 'auth',
    keyXMLDebuggingAttr = 'xdbg',
    kAERPCClass = 'rpc ',
    kAEXMLRPCScheme = 'RPC2',
    kAESOAPScheme = 'SOAP',
    kAESharedScriptHandler = 'wscp',
    keyRPCMethodName = 'meth',
    keyRPCMethodParam = 'parm',
    keyRPCMethodParamOrder = '/ord',
    keyAEPOSTHeaderData = 'phed',
    keyAEReplyHeaderData = 'rhed',
    keyAEXMLRequestData = 'xreq',
    keyAEXMLReplyData = 'xrep',
    keyAdditionalHTTPHeaders = 'ahed',
    keySOAPAction = 'sact',
    keySOAPMethodNameSpace = 'mspc',
    keySOAPMethodNameSpaceURI = 'mspu',
    keySOAPSchemaVersion = 'ssch'
};
```

kFAServerApp

```
enum {
    kFAServerApp = 'ssrv',
    kDoFolderActionEvent = 'fola',
    kFolderActionCode = 'actn',
    kFolderOpenedEvent = 'fopn',
    kFolderClosedEvent = 'fclo',
    kFolderWindowMovedEvent = 'fsiz',
    kFolderItemsAddedEvent = 'fget',
    kFolderItemsRemovedEvent = 'flos',
    kItemList = 'flst',
    kNewSizeParameter = 'fnsz',
    kFASuiteCode = 'faco',
    kFAAttachCommand = 'atfa',
```

```
    kFARemoveCommand = 'rmfa',
    kFAEditCommand = 'edfa',
    kFAFileParam = 'faal',
    kFAIndexParam = 'indx'
};
```

kLaunchToGetTerminology

```
enum {
    kLaunchToGetTerminology = 0x8000,
    kDontFindAppBySignature = 0x4000,
    kAlwaysSendSubject = 0x2000
};
```

kNextBody

```
enum {
    kNextBody = 1,
    kPreviousBody = 2
};
```

kOSIZDontOpenResourceFile

```
enum {
    kOSIZDontOpenResourceFile = 15,
    kOSIZdontAcceptRemoteEvents = 14,
    kOSIZOpenWithReadPermission = 13,
    kOSIZCodeInSharedLibraries = 11
};
```

kReadExtensionTermsMask

```
enum {
    kReadExtensionTermsMask = 0x8000
};
```

kSOAP1999Schema

```
enum {
    kSOAP1999Schema = 'ss99',
    kSOAP2001Schema = 'ss01'
};
```

kTextServiceClass

```
enum {
    kTextServiceClass = 'tsvc',
    kUpdateActiveInputArea = 'updt',
```

```
    kShowHideInputWindow = 'shiw',
    kPos2Offset = 'p2st',
    kOffset2Pos = 'st2p',
    kUnicodeNotFromInputMethod = 'unim',
    kGetSelectedText = 'gtxt',
    keyAETSMDocumentRefcon = 'refc',
    keyAEServerInstance = 'srvi',
    keyAETheData = 'kdat',
    keyAEFixLength = 'fixl',
    keyAEUpdateRange = 'udng',
    keyAECurrentPoint = 'cpos',
    keyAEBufferSize = 'buff',
    keyAEMoveView = 'mvvw',
    keyAENextBody = 'nxbd',
    keyAETSMScriptTag = 'sclg',
    keyAETSMTextFont = 'ktxf',
    keyAETSMTextFMFont = 'ktxm',
    keyAETSMTextPointSize = 'ktps',
    keyAETSMEventRecord = 'tevt',
    keyAETSMEventRef = 'tevr',
    keyAETextServiceEncoding = 'tsen',
    keyAETextServiceMacEncoding = 'tmen',
    keyAETSMGlyphInfoArray = 'tgia',
    typeTextRange = 'txrn',
    typeComponentInstance = 'cmpi',
    typeOffsetArray = 'ofay',
    typeTextRangeArray = 'tray',
    typeLowLevelEventRecord = 'evtr',
    typeGlyphInfoArray = 'glia',
    typeEventRef = 'evrf',
    typeText = 'TEXT'
};
```

kTSMHiliteCaretPosition

Specify text highlighting information.

```
enum {
    kTSMHiliteCaretPosition = 1,
    kTSMHiliteRawText = 2,
    kTSMHiliteSelectedRawText = 3,
    kTSMHiliteConvertedText = 4,
    kTSMHiliteSelectedConvertedText = 5,
    kTSMHiliteBlockFillText = 6,
    kTSMHiliteOutlineText = 7,
    kTSMHiliteSelectedText = 8,
    kTSMHiliteNoHilite = 9
};
```

Constants

`kTSMHiliteCaretPosition`

Specifies caret position.

Available in Mac OS X v10.4 and later.

Declared in `AERegistry.h`.

`kTSMHiliteRawText`
Specifies range of raw text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteSelectedRawText`
Specifies range of selected raw text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteConvertedText`
Specifies range of converted text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteSelectedConvertedText`
Specifies range of selected converted text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteBlockFillText`
Specifies block fill highlight style.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteOutlineText`
Specifies outline highlight style.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteSelectedText`
Specifies selected highlight style.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteNoHilite`
Specifies range of non-highlighted text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

Version Notes
This enumeration is available starting in Mac OS X v10.4. Use these constants in place of the constants defined in "kCaretPosition" (page 615).

kTSMOutsideOfBody

```
enum {
    kTSMOutsideOfBody = 1,
    kTSMInsideOfBody = 2,
    kTSMInsideOfActiveInputArea = 3
};
```

pArcAngle

```
enum {
    pArcAngle = 'parc',
    pBackgroundColor = 'pbcl',
    pBackgroundPattern = 'pbpt',
    pBestType = 'pbst',
    pBounds = 'pbnd',
    pClass = 'pcls',
    pClipboard = 'pcli',
    pColor = 'colr',
    pColorTable = 'cltb',
    pContents = 'pcnt',
    pCornerCurveHeight = 'pchd',
    pCornerCurveWidth = 'pcwd',
    pDashStyle = 'pdst',
    pDefaultType = 'deft',
    pDefinitionRect = 'pdrt',
    pEnabled = 'enbl',
    pEndPoint = 'pend',
    pFillColor = 'flcl',
    pFillPattern = 'flpt',
    pFont = 'font'
};
```

pFormula

```
enum {
    pFormula = 'pfor',
    pGraphicObjects = 'gobs',
    pHasCloseBox = 'hclb',
    pHasTitleBar = 'ptit',
    pID = 'ID  ',
    pIndex = 'pidx',
    pInsertionLoc = 'pins',
    pIsFloating = 'isfl',
    pIsFrontProcess = 'pisf',
    pIsModal = 'pmod',
    pIsModified = 'imod',
    pIsResizable = 'prsz',
    pIsStationeryPad = 'pspd',
    pIsZoomable = 'iszm',
    pIsZoomed = 'pzum',
    pItemNumber = 'itmn',
    pJustification = 'pjst',
    pLineArrow = 'arro',
    pMenuID = 'mnid',
```

```
    pName = 'pnam'
};
```

pNewElementLoc

```
enum {
    pNewElementLoc = 'pnel',
    pPenColor = 'ppcl',
    pPenPattern = 'pppa',
    pPenWidth = 'ppwd',
    pPixelDepth = 'pdpt',
    pPointList = 'ptlt',
    pPointSize = 'ptsz',
    pProtection = 'ppro',
    pRotation = 'prot',
    pScale = 'pscl',
    pScript = 'scpt',
    pScriptTag = 'psct',
    pSelected = 'selc',
    pSelection = 'sele',
    pStartAngle = 'pang',
    pStartPoint = 'pstp',
    pTextColor = 'ptxc',
    pTextFont = 'ptxf',
    pTextItemDelimiters = 'txdl',
    pTextPointSize = 'ptps'
};
```

pScheme

```
enum {
    pScheme = 'pusc',
    pHost = 'HOST',
    pPath = 'FTPc',
    pUserName = 'RAun',
    pUserPassword = 'RApw',
    pDNSForm = 'pDNS',
    pURL = 'pURL',
    pTextEncoding = 'ptxe',
    pFTPKind = 'kind'
};
```

pTextStyles

```
enum {
    pTextStyles = 'txst',
    pTransferMode = 'pptm',
    pTranslation = 'ptrs',
    pUniformStyles = 'ustl',
    pUpdateOn = 'pupd',
    pUserSelection = 'pusl',
    pVersion = 'vers',
    pVisible = 'pvis'
```

```
};
```

typeAEText

```
enum {
    typeAEText = 'tTXT',
    typeArc = 'carc',
    typeBest = 'best',
    typeCell = 'ccel',
    typeClassInfo = 'gcli',
    typeColorTable = 'clrt',
    typeColumn = 'ccol',
    typeDashStyle = 'tdas',
    typeData = 'tdta',
    typeDrawingArea = 'cdrw',
    typeElemInfo = 'elin',
    typeEnumeration = 'enum',
    typeEPS = 'EPS ',
    typeEventInfo = 'evin'
};
```

typeApplicationBundleID

For specifying a target application by bundle ID.

```
enum {
    typeApplicationBundleID = 'bund'
};
```

Constants

`typeApplicationBundleID`

Indicates a descriptor containing UTF-8 characters that specify the bundle ID of an application. Bundle IDs should be constructed similarly to `"com.company.directorylocation.ApplicationName"`.

Available in Mac OS X v10.3 and later.

Declared in `AEDataModel.h`.

Discussion

This address mode is preferred for targeting specific applications. For example, you should target the Finder by sending an event whose target address descriptor uses the bundle ID `"com.apple.finder"` rather than the application signature `'MACS'`.

typeFinderWindow

```
enum {
    typeFinderWindow = 'fwin',
    typeFixedPoint = 'fpnt',
    typeFixedRectangle = 'frct',
    typeGraphicLine = 'glin',
    typeGraphicText = 'cgtx',
    typeGroupedGraphic = 'cpic',
    typeInsertionLoc = 'insl',
    typeIntlText = 'itxt',
    typeIntlWritingCode = 'intl',
    typeLongDateTime = 'ldt ',
    typeISO8601DateTime = 'isot',
    typeLongFixed = 'lfxd',
    typeLongFixedPoint = 'lfpt',
    typeLongFixedRectangle = 'lfrc',
    typeLongPoint = 'lpnt',
    typeLongRectangle = 'lrct',
    typeMachineLoc = 'mLoc',
    typeOval = 'covl',
    typeParamInfo = 'pmin',
    typePict = 'PICT'
};
```

Constants

`typeIntlText`

For important information, see the Version Notes section of the "`typeUnicodeText`" (page 635) enum.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

typeHIMenu

```
enum {
    typeHIMenu = 'mobj',
    typeHIWindow = 'wobj'
};
```

typeKernelProcessID

For specifying an application by UNIX process ID.

```
enum {
    typeKernelProcessID = 'kpid'
};
```

Constants

`typeKernelProcessID`

Indicates a descriptor containing a UNIX process ID. A process ID is similar to a PSN (processor serial number) but does not require a Process Manager connection. It is analogous to a 32-bit unsigned integer.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

You might use this constant in a situation where you have access to the PID for a process but don't have a Process Manager connection. Your code for creating the descriptor might look like the following:

```
pid_t pid = findTheAppPid(); // User supplied routine to get PID. // Now create
 a descriptor with it: AECreateDesc(typeKernelProcessID, &pid, sizeof(pid),
&desc);
```

typeMachPort

For specifying a Mach port.

```
enum {
    typeMachPort = 'port'
};
```

Constants

`typeMachPort`

Indicates a descriptor that specifies a Mach port.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

You might use this constant as part of sending an Apple event to an arbitrary Mach port. Your code for creating the descriptor might look like the following:

```
mach_port_t port = lookupPortForTarget(); // User routine to get  port.
// Now create a descriptor with it:
AECreateDesc(typeMachPort, &port, sizeof(port), &desc);
```

Actually sending an Apple event to a Mach port is an advanced technique and is not documented here.

typeMeters

```
enum {
    typeMeters = 'metr',
    typeInches = 'inch',
    typeFeet = 'feet',
    typeYards = 'yard',
    typeMiles = 'mile',
    typeKilometers = 'kmtr',
    typeCentimeters = 'cmtr',
    typeSquareMeters = 'sqrm',
    typeSquareFeet = 'sqft',
    typeSquareYards = 'sqyd',
    typeSquareMiles = 'sqmi',
    typeSquareKilometers = 'sqkm',
    typeLiters = 'litr',
    typeQuarts = 'qrts',
    typeGallons = 'galn',
    typeCubicMeters = 'cmet',
    typeCubicFeet = 'cfet',
    typeCubicInches = 'cuin',
    typeCubicCentimeter = 'ccmt',
    typeCubicYards = 'cyrd',
    typeKilograms = 'kgrm',
    typeGrams = 'gram',
    typeOunces = 'ozs ',
    typePounds = 'lbs ',
    typeDegreesC = 'degc',
    typeDegreesF = 'degf',
    typeDegreesK = 'degk'
};
```

typePixelMap

```
enum {
    typePixelMap = 'cpix',
    typePixMapMinus = 'tpmm',
    typePolygon = 'cpgn',
    typePropInfo = 'pinf',
    typePtr = 'ptr ',
    typeQDPoint = 'QDpt',
    typeQDRegion = 'Qrgn',
    typeRectangle = 'crec',
    typeRGB16 = 'tr16',
    typeRGB96 = 'tr96',
    typeRGBColor = 'cRGB',
    typeRotation = 'trot',
    typeRoundedRectangle = 'crrc',
    typeRow = 'crow',
    typeScrapStyles = 'styl',
    typeScript = 'scpt',
    typeStyledText = 'STXT',
    typeSuiteInfo = 'suin',
    typeTable = 'ctbl',
    typeTextStyles = 'tsty'
};
```

Constants

`typeStyledText`

Text that includes style information.

Styled text is stored as a record, in which the styles have the key `'ksty'` and the plain text is has the key `'ktxt'`. You can use this information to extract plain text from styled text without coercion.

However, getting rid of the style information, with or without coercion, may corrupt the text, since the styles imply what encoding to use. In fact, use of `typeText` and `typeStyledText` are not recommended, starting with Mac OS X, because they are not safe with international characters—you should use one of the Unicode text types instead.

For important information, see the Version Notes section of the "`typeUnicodeText`" (page 635) enum.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

typeReplyPortAttr

```
enum {
    typeReplyPortAttr = 'repp'
};
```

typeSessionID

```
enum {
    typeSessionID = 'ssid',
    typeTargetID = 'targ',
    typeDispatcherID = 'dspt'
};
```

Constants

`typeSessionID`

Session reference number.

`typeTargetID`

Target ID descriptor. Target IDs are not supported in Mac OS X.

typeSMInt

Where possible, you should use the constants defined in "Numeric Descriptor Type Constants" (page 597), rather than those defined here.

```
enum {
    typeSMInt = 'shor',
    typeShortInteger = 'shor',
    typeInteger = 'long',
    typeLongInteger = 'long',
    typeMagnitude = 'magn',
    typeComp = 'comp',
    typeSMFloat = 'sing',
    typeShortFloat = 'sing',
    typeFloat = 'doub',
    typeLongFloat = 'doub',
    typeExtended = 'exte'
};
```

Constants

`typeSMInt`
16-bit integer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeShortInteger`
16-bit integer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeInteger`
32-bit integer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeLongInteger`
32-bit integer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeMagnitude`
Unsigned 32-bit integer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeComp`
Standard Apple Numerics Environment (SANE) comparison operator.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeSMFloat`
SANE single.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeShortFloat`
SANE single.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeFloat`
SANE double.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeLongFloat`
SANE double.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeExtended`
SANE extended.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

typeTIFF

```
enum {
    typeTIFF = 'TIFF',
    typeVersion = 'vers'
};
```

typeUnicodeText

```
enum {
    typeUTF16ExternalRepresentation = 'ut16',
    typeUnicodeText = 'utxt',
    typeStyledUnicodeText = 'sutx',
    typeUTF8Text = 'utf8',
    typeEncodedString = 'encs',
    typeCString = 'cstr',
    typePString = 'pstr'
};
```

Constants

`typeUTF16ExternalRepresentation`
Unicode text in 16-bit external representation with byte-order-mark (BOM).
Guarantees that either there is a BOM or the data is in UTF-16BE.
Available in Mac OS X v10.4 and later.
Declared in `AEDataModel.h`.

`typeUnicodeText`
Unicode text. Native byte ordering, optional BOM.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeStyledUnicodeText`
Styled Unicode text. Not implemented.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeUTF8Text`
8-bit Unicode (UTF-8 encoding).
Available in Mac OS X v10.2 and later.
Declared in `AEDataModel.h`.

`typeEncodedString`
Styled Unicode text. Not implemented.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeCString`
C string—Mac OS Roman characters followed by a NULL byte. Deprecated.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typePString`

Pascal string—unsigned length byte followed by Mac OS Roman characters. Deprecated.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Version Notes
In Mac OS X version 10.4, you should use `typeUTF16ExternalRepresentation` or `typeUTF8Text` to represent text. In earlier versions of Mac OS X, the recommended text type is `typeUnicodeText`. All of the other constants in this enum are deprecated due to their lack of explicit encoding or byte order definition.

The implicitly encoded text types, `typeText`, `typeCString`, and `typePString`, are all deprecated in Mac OS X, because they are incapable of representing international characters and may be reinterpreted in unpredictable ways. Additionally, `typeCString` and `typePString` do not support the full range of text coercions, and will be removed entirely in a future release. `typeStyledText` and `typeIntlText`, while they have explicit encodings, are not recommended, since they are incapable of representing Unicode-only characters, such as Hungarian, Arabic, or Thai.

Result Codes

Because the Apple Event Manager uses the services of the Event Manager, the functions described in this document may return Event Manager result codes in addition to the Apple Event Manager result codes listed here. Less commonly, an Apple Event Manager function may return other result codes, including some of those found in the CarbonCore header file `MacErrors.h`.

For result codes for the AEBuild-related functions, see "AEBuild Error Codes" (page 563).

Result Code	Value	Description
`noPortErr`	-903	Client hasn't set `'SIZE'` resource to indicate awareness of high-level events Available in Mac OS X v10.0 and later.
`destPortErr`	-906	Server hasn't set `'SIZE'` resource to indicate awareness of high-level events, or else is not present Available in Mac OS X v10.0 and later.
`sessClosedErr`	-917	The `kAEDontReconnect` flag in the `sendMode` parameter was set and the server quit, then restarted Available in Mac OS X v10.0 and later.
`errAECoercionFail`	-1700	Data could not be coerced to the requested descriptor type Available in Mac OS X v10.0 and later.
`errAEDescNotFound`	-1701	Descriptor was not found Available in Mac OS X v10.0 and later.
`errAECorruptData`	-1702	Data in an Apple event could not be read Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errAEWrongDataType	-1703	Wrong descriptor type Available in Mac OS X v10.0 and later.
errAENotAEDesc	-1704	Not a valid descriptor Available in Mac OS X v10.0 and later.
errAEBadListItem	-1705	Operation involving a list item failed Available in Mac OS X v10.0 and later.
errAENewerVersion	-1706	Need a newer version of the Apple Event Manager Available in Mac OS X v10.0 and later.
errAENotAppleEvent	-1707	The event is not in AppleEvent format. Available in Mac OS X v10.0 and later.
errAEEventNotHandled	-1708	Event wasn't handled by an Apple event handler Available in Mac OS X v10.0 and later.
errAEReplyNotValid	-1709	AEResetTimer was passed an invalid reply Available in Mac OS X v10.0 and later.
errAEUnknownSendMode	-1710	Invalid sending mode was passed Available in Mac OS X v10.0 and later.
errAEWaitCanceled	-1711	User canceled out of wait loop for reply or receipt Available in Mac OS X v10.0 and later.
errAETimeout	-1712	Apple event timed out Available in Mac OS X v10.0 and later.
errAENoUserInteraction	-1713	No user interaction allowed Available in Mac OS X v10.0 and later.
errAENotASpecialFunction	-1714	Wrong keyword for a special function Available in Mac OS X v10.0 and later.
errAEParamMissed	-1715	A required parameter was not accessed. Available in Mac OS X v10.0 and later.
errAEUnknownAddressType	-1716	Unknown Apple event address type Available in Mac OS X v10.0 and later.
errAEHandlerNotFound	-1717	No handler found for an Apple event Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errAEReplyNotArrived	-1718	Reply has not yet arrived Available in Mac OS X v10.0 and later.
errAEIllegalIndex	-1719	Not a valid list index Available in Mac OS X v10.0 and later.
errAEImpossibleRange	-1720	The range is not valid because it is impossible for a range to include the first and last objects that were specified; an example is a range in which the offset of the first object is greater than the offset of the last object Available in Mac OS X v10.0 and later.
errAEWrongNumberArgs	-1721	The number of operands provided for the kAENOT logical operator is not 1 Available in Mac OS X v10.0 and later.
errAEAccessorNotFound	-1723	There is no object accessor function for the specified object class and container type Available in Mac OS X v10.0 and later.
errAENoSuchLogical	-1725	The logical operator in a logical descriptor is not kAEAND, kAEOR, or kAENOT Available in Mac OS X v10.0 and later.
errAEBadTestKey	-1726	The descriptor in a test key is neither a comparison descriptor nor a logical descriptor Available in Mac OS X v10.0 and later.
errAENotAnObjectSpec	–1727	The objSpecifier parameter of AEResolve is not an object specifier
errAENoSuchObject	-1728	Runtime resolution of an object failed. Available in Mac OS X v10.0 and later.
errAENegativeCount	-1729	An object-counting function returned a negative result Available in Mac OS X v10.0 and later.
errAEEmptyListContainer	-1730	The container for an Apple event object is specified by an empty list Available in Mac OS X v10.0 and later.
errAEUnknownObjectType	-1731	The object type isn't recognized Available in Mac OS X v10.0 and later.
errAERecordingIsAlreadyOn	-1732	Recording is already on Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errAEReceiveTerminate	-1733	Break out of all levels of AEReceive to the topmost (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEReceiveEscapeCurrent	-1734	Break out of lowest level only of AEReceive (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEEventFiltered	-1735	Event has been filtered and should not be propagated (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEDuplicateHandler	-1736	Attempt to install handler in table for identical class and ID (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEStreamBadNesting	-1737	Nesting violation while streaming Available in Mac OS X v10.0 and later.
errAEStreamAlreadyConverted	-1738	Attempt to convert a stream that has already been converted Available in Mac OS X v10.0 and later.
errAEDescIsNull	-1739	Attempt to perform an invalid operation on a null descriptor Available in Mac OS X v10.0 and later.
errAEBuildSyntaxError	-1740	AEBuildDesc and related functions detected a syntax error Available in Mac OS X v10.0 and later.
errAEBufferTooSmall	-1741	Buffer for AEFlattenDesc too small Available in Mac OS X v10.0 and later.
errASCantConsiderAndIgnore	-2720	Can't both consider and ignore <attribute>. Available in Mac OS X v10.0 and later.
errASCantCompareMoreThan32k	-2721	Can't perform operation on text longer than 32K bytes. Available in Mac OS X v10.0 and later.
errASTerminologyNestingTooDeep	-2760	Tell statements are nested too deeply. Available in Mac OS X v10.0 and later.
errASIllegalFormalParameter	-2761	<name> is illegal as a formal parameter. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errASParameterNotForEvent	-2762	<name> is not a parameter name for the event <event>. Available in Mac OS X v10.0 and later.
errASNoResultReturned	-2763	No result was returned for some argument of this expression. Available in Mac OS X v10.0 and later.
errAEEventFailed	-10000	Apple event handler failed. Available in Mac OS X v10.0 and later.
errAETypeError	-10001	A descriptor type mismatch occurred. Available in Mac OS X v10.0 and later.
errAEBadKeyForm	-10002	Invalid key form. Available in Mac OS X v10.0 and later.
errAENotModifiable	-10003	Can't set <object or data> to <object or data>. Access not allowed. Available in Mac OS X v10.0 and later.
errAEPrivilegeError	-10004	A privilege violation occurred. Available in Mac OS X v10.0 and later.
errAEReadDenied	-10005	The read operation was not allowed. Available in Mac OS X v10.0 and later.
errAEWriteDenied	-10006	Can't set <object or data> to <object or data>. Available in Mac OS X v10.0 and later.
errAEIndexTooLarge	-10007	The index of the event is too large to be valid. Available in Mac OS X v10.0 and later.
errAENotAnElement	-10008	The specified object is a property, not an element. Available in Mac OS X v10.0 and later.
errAECantSupplyType	-10009	Can’t supply the requested descriptor type for the data. Available in Mac OS X v10.0 and later.
errAECantHandleClass	-10010	The Apple event handler can’t handle objects of this class. Available in Mac OS X v10.0 and later.
errAEInTransaction	-10011	Couldn’t handle this command because it wasn’t part of the current transaction. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`errAENoSuchTransaction`	-10012	The transaction to which this command belonged isn't a valid transaction. Available in Mac OS X v10.0 and later.
`errAENoUserSelection`	-10013	There is no user selection. Available in Mac OS X v10.0 and later.
`errAENotASingleObject`	-10014	Handler only handles single objects. Available in Mac OS X v10.0 and later.
`errAECantUndo`	-10015	Can't undo the previous Apple event or user action. Available in Mac OS X v10.0 and later.
`errAENotAnEnumMember`	-10023	Enumerated value in `SetData` is not allowed for this property Available in Mac OS X v10.0 and later.
`errAECantPutThatThere`	-10024	In make new, duplicate, etc. class can't be an element of container Available in Mac OS X v10.0 and later.
`errAEPropertiesClash`	-10025	Illegal combination of properties settings for SetData, make new, or duplicate Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Apple Event Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Apple Type Services for Fonts Reference

Framework: ApplicationServices/ApplicationServices.h

Declared in ATSTypes.h
ATSFont.h

Overview

Apple Type Services for Fonts is a collection of functions and data types that you can use to access and manage font data in Mac OS X. It is designed to handle a wide range of font technologies and data formats. The programming interface is designed with performance, scalability, and consistency in mind, and is available to Cocoa and Carbon applications through the Apple Type Services (ATS) and QuickDraw frameworks in Mac OS X.

Carbon supports the Apple Type Services for Fonts.

Functions by Task

Activating and Deactivating Fonts

`ATSFontActivateFromFileReference` (page 647)
Activates one or more fonts from a file reference.

`ATSFontActivateFromMemory` (page 649)
Activates one or more fonts at the specified location in memory.

`ATSFontDeactivate` (page 651)
Deactivates one or more fonts.

`ATSGetGeneration` (page 678)
Obtains the generation of the font database.

`ATSFontSetGlobalAutoActivationSetting` (page 678)
Sets the user's global auto-activation setting.

`ATSFontGetGlobalAutoActivationSetting` (page 666)
Gets the user's global auto-activation setting.

`ATSFontSetAutoActivationSettingForApplication` (page 677)
Sets the auto-activation setting for the specified application bundle.

`ATSFontGetAutoActivationSettingForApplication` (page 661)
Gets the activation setting for the specified application.

ATSFontActivateFromFileSpecification (page 648) Deprecated in Mac OS X v10.5
Activates one or more fonts from a file specification. (Deprecated. Instead use ATSFontActivateFromFileReference (page 647).)

Working With Font Families

ATSFontFamilyApplyFunction (page 652)
Applies your callback to a font family iteration.

ATSFontFamilyIteratorCreate (page 655)
Creates a font family iterator that your application can use to access font family objects.

ATSFontFamilyIteratorRelease (page 658)
Releases the memory associated with a font family iterator.

ATSFontFamilyIteratorReset (page 658)
Resets a font family iterator to the beginning of the iteration.

ATSFontFamilyIteratorNext (page 657)
Obtains the next font family reference.

ATSFontFamilyFindFromName (page 652)
Returns the font family reference associated with a font family name.

ATSFontFamilyFindFromQuickDrawName (page 653)
Returns the font family reference associated with a standard QuickDraw font name.

ATSFontFamilyGetGeneration (page 654)
Returns the generation count of a font family.

ATSFontFamilyGetName (page 654)
Obtains the font family name associated with a font family reference.

ATSFontFamilyGetQuickDrawName (page 655)
Obtains the standard QuickDraw font name associated with a font family reference.

ATSFontFamilyGetEncoding (page 653)
Returns the text encoding used by a font family.

Working With Fonts

ATSFontApplyFunction (page 650)
Applies your callback to a font iteration.

ATSFontIteratorCreate (page 671)
Creates a font iterator.

ATSFontIteratorRelease (page 673)
Releases a font iterator.

ATSFontIteratorReset (page 674)
Resets a font iterator to the beginning of the iteration.

ATSFontIteratorNext (page 672)
Obtains the next font reference.

ATSFontFindFromName (page 660)
Returns the font reference associated with a font name.

`ATSFontFindFromPostScriptName` (page 661)
Returns the font reference associated with a PostScript font name.

`ATSFontFindFromContainer` (page 659)
Obtains the font references contained in a font container.

`ATSFontGetGeneration` (page 665)
Returns the generation count for a font.

`ATSFontGetContainerFromFileReference` (page 662)
Gets the font container reference associated with an activated file reference.

`ATSFontGetContainer` (page 662)
Gets the font container reference for a font.

`ATSFontGetName` (page 666)
Obtains the name of a font associated with a font reference.

`ATSFontGetPostScriptName` (page 667)
Obtains the PostScript name from a font reference.

`ATSFontGetTableDirectory` (page 669)
Obtains the table directory for a font.

`ATSFontGetTable` (page 668)
Obtains a font table.

`ATSFontGetHorizontalMetrics` (page 666)
Obtains the horizontal metrics for a font.

`ATSFontGetVerticalMetrics` (page 670)
Obtains the vertical metrics for a font.

`ATSFontGetFileReference` (page 663)
Obtains the file reference for a font.

`ATSFontGetFontFamilyResource` (page 664)
Obtains the font family resource for a font.

`ATSFontSetEnabled` (page 678)
Sets a font state to enabled or disabled.

`ATSFontIsEnabled` (page 671)
Returns `true` if the font is enabled.

`ATSFontGetFileSpecification` (page 663) Deprecated in Mac OS X v10.5
Obtains the file specification for a font. (Deprecated. Instead use `ATSFontGetFileReference` (page 663).)

Setting Up Notifications and Queries

`ATSFontNotify` (page 676)
Notifies Apple Type Services of an action taken by your application.

`ATSFontNotificationSubscribe` (page 675)
Signs up your application to receive notification of changes to fonts and font directories.

`ATSFontNotificationUnsubscribe` (page 676)
Unsubscribes your application from receiving notifications of changes to fonts and font directories.

`ATSCreateFontQueryRunLoopSource` (page 646)
Sets up your application to handle font queries.

Creating, Calling, and Deleting Universal Procedure Pointers

`NewFMFontCallbackFilterUPP` (page 680)
Creates a new universal procedure pointer (UPP) to a filter callback function that uses your criteria for filtering fonts.

`DisposeFMFontCallbackFilterUPP` (page 679)
Disposes of a universal procedure pointer to a customized filter function used for fonts.

`InvokeFMFontCallbackFilterUPP` (page 680)
Calls a customized filter function used for fonts.

`NewFMFontFamilyCallbackFilterUPP` (page 681)
Creates a new universal procedure pointer (UPP) to a filter callback function that uses your criteria for filtering font families.

`DisposeFMFontFamilyCallbackFilterUPP` (page 679)
Disposes of a universal procedure pointer to a customized filter function used for font families.

`InvokeFMFontFamilyCallbackFilterUPP` (page 680)
Calls a customized filter function used for font families.

Functions

ATSCreateFontQueryRunLoopSource

Sets up your application to handle font queries.

```
CFRunLoopSourceRef ATSCreateFontQueryRunLoopSource (
   CFIndex queryOrder,
   CFIndex sourceOrder,
   ATSFontQueryCallback callout,
   const ATSFontQuerySourceContext *context
);
```

Parameters

queryOrder
A `CFIndex` value that specifies the priority of the query relative to other queries. ATS sends font queries to each run loop in priority order, from highest to lowest, with normal priority equal to `0`.

sourceOrder
A `CFIndex` value that specifies the order of the run loop source.

callout
A pointer to your callback for processing a font query. See `ATSFontQueryCallback` (page 683) for more information on the callback you can supply.

context
A pointer to font query source context that ATS passes to your callback function. You can pass `NULL` if your callback function does not need any data passed to it.

Return Value
A `CFRunLoopSourceRef`. When you want to stop receiving queries, you must invalidate this reference.

Discussion
When an application needs a font, ATS sends a query to those font utility applications who have signed up to handle queries by calling the function `ATSCreateFontQueryRunLoopSource`. When a font utility application receives a query, it iterates through its available fonts to look for the requested font. If the font utility application finds the font, it obtains the file specification for the font and sends the file specification to ATS. ATS activates the font and sends notification of the activation to each application who subscribes to notifications.

The function `ATSCreateFontQueryRunLoopSource` creates a Core Foundation run loop source reference (`CFRunLoopSourceRef`) to convey font queries from ATS to your font utility application. If your application does not have a `CFRunLoop` (for example, a faceless server application), you must explicitly set up a `CFRunLoop` before you can receive queries.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.2 and later.

Declared In
`ATSFont.h`

ATSFontActivateFromFileReference

Activates one or more fonts from a file reference.

```
OSStatus ATSFontActivateFromFileReference (
   const FSRef *iFile,
   ATSFontContext iContext,
   ATSFontFormat iFormat,
   void *iRefCon,
   ATSOptionFlags iOptions,
   ATSFontContainerRef *oContainer
);
```

Parameters

iFile

A pointer to the file reference that specifies the name and location of a file or directory that contains the font data you want to activate.

iContext

A value that specifies the context of the activated font. If you want the activated font to be accessible only from your application use the `kATSFontContextLocal` constant. If you want the activated font to be accessible to all applications use the constant `kATSFontContextGlobal`. See "Context Options" (page 703) for more information.

iFormat

A value that represents the format identifier of the font. Pass `kATSFontFormatUnspecified` as the system automatically determines the format of the font. For more information on this constant, see "Font Formats" (page 706).

iRefCon

This parameter is currently reserved for future use, so you should pass `NULL`.

iOptions

An options flag. Pass `kATSOptionFlagsDefault` unless the font's data fork contains resource-fork information, you need to activate a directory of font directories, or you plan to call this function a number of time. If the font's data fork contains resource-fork information, pass the option `kATSOptionFlagsUseDataForkAsResourceFork`. If the you want to activate a font directory that contains font directories, you must pass the option `kATSOptionFlagsProcessSubdirectories`. If you plan to call this function a number of times, you can set the `iOptions` parameter to `kATSOptionFlagsDoNotNotify` set. When you are done activating fonts you can call the function `ATSFontNotify` (page 676) with the `action` parameter set to `kATSFontNotifyActionFontsChanged`. Then ATS notifies all applications who subscribe to notifications of the changes you made.

oContainer

On output, a reference to the font container that is activated from the file reference. You need this reference when you deactivate the font by calling the function `ATSFontDeactivate` (page 651).

Return Value
If activated successfully, `noErr`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`ATSFont.h`

ATSFontActivateFromFileSpecification

Activates one or more fonts from a file specification. (Deprecated in Mac OS X v10.5. Instead use `ATSFontActivateFromFileReference` (page 647).)

```
OSStatus ATSFontActivateFromFileSpecification (
   const FSSpec *iFile,
   ATSFontContext iContext,
   ATSFontFormat iFormat,
   void *iReserved,
   ATSOptionFlags iOptions,
   ATSFontContainerRef *oContainer
);
```

Parameters

iFile

A pointer to the file specification that specifies the name and location of a file or directory that contains the font data you want to activate.

iContext

A value that specifies the context of the activated font. If you want the activated font to be accessible only from your application use the `kATSFontContextLocal` constant. If you want the activated font to be accessible to all applications use the constant `kATSFontContextGlobal`. See "Context Options" (page 703) for more information.

iFormat

A value that represents the format identifier of the font. Pass `kATSFontFormatUnspecified` as the system automatically determines the format of the font. For more information on this constant, see "Font Formats" (page 706).

iReserved

An arbitrary 32-bit value. This parameter is currently reserved for future use, so you should pass `NULL`.

iOptions

An options flag. Pass `kATSOptionFlagsDefault` unless the font's data fork contains resource-fork information, you need to activate a directory of font directories, or you plan to call this function a number of time. If the font's data fork contains resource-fork information, pass the option `kATSOptionFlagsUseDataForkAsResourceFork`. If the you want to activate a font directory that contains font directories, you must pass the option `kATSOptionFlagsProcessSubdirectories`. If you plan to call this function a number of times, you can set the `iOptions` parameter to `kATSOptionFlagsDoNotNotify` set. When you are done activating fonts you can call the function `ATSFontNotify` (page 676) with the `action` parameter set to `kATSFontNotifyActionFontsChanged`. Then ATS notifies all applications who subscribe to notifications of the changes you made.

oContainer

On output, a reference to the font container that is activated from the file specification. You need this reference when you deactivate the font by calling the function `ATSFontDeactivate` (page 651).

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
You can use the function `ATSFontActivateFromFileSpecification` to activate one font or more fonts. Activating a font makes that font available for use either locally (available only to your application) or globally (available to all applications on the system). A font's availability—local or global—is referred to as its context.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`ATSFont.h`

ATSFontActivateFromMemory

Activates one or more fonts at the specified location in memory.

```
OSStatus ATSFontActivateFromMemory (
   LogicalAddress iData,
   ByteCount iLength,
   ATSFontContext iContext,
   ATSFontFormat iFormat,
   void *iReserved,
   ATSOptionFlags iOptions,
   ATSFontContainerRef *oContainer
);
```

Parameters

iData

The logical address of the font you want to activate.

iLength
The length (in bytes) of the font data.

iContext
A value that specifies the context of the activated font. If you want the activated font to be accessible only from your application use the `kATSFontContextLocal` constant. If you want the activated font to be accessible to all applications use the constant `kATSFontContextGlobal`. See "Context Options" (page 703) for more information.

iFormat
A value that represents the format identifier of the font. There is only one font format constant available for you to pass—`kATSFontFormatUnspecified`. This constant specifies the default behavior, which is to handle the data as raw TrueType font data. This is equivalent to the contents of an `'sfnt'` font resource or the data fork of a Windows TrueType `.ttf` or `.ttc` file. You can also activate the contents of an OpenType TrueType `.OTF` file. See "Font Formats" (page 706) for more information.

iReserved
An arbitrary 32-bit value. This parameter is currently reserved for future use, so you should pass `NULL`.

iOptions
An `ATSOptionFlags` value. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

oContainer
On output, a pointer to a font container reference that refers to the file that contains the activated font data.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
You use this function to activate a streamed font, such as a font contained in a PDF file. Your application must first map the streamed font data to memory and then pass the address of the font data in memory to the function `ATSFontActivateFromMemory`.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontApplyFunction

Applies your callback to a font iteration.

```
OSStatus ATSFontApplyFunction (
   ATSFontApplierFunction iFunction,
   void *iRefCon
);
```

Parameters

iFunction
The callback function you want applied to a font iteration. See `ATSFontApplierFunction` (page 682) for more information on the callback you need to supply.

iRefCon

An arbitrary 32-bit value specified by your application. This is passed to your callback.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
The function `ATSFontApplyFunction` iterates through the default fonts, which include globally activated fonts and fonts activated locally to your application. Calling this function is similar to creating an iterator that operates on a local context with an unrestricted scope.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontDeactivate

Deactivates one or more fonts.

```
OSStatus ATSFontDeactivate (
   ATSFontContainerRef iContainer,
   void *iRefCon,
   ATSOptionFlags iOptions
);
```

Parameters

iContainer

A font container reference that refers to the file containing the activated font data. You obtain a font container reference when you activate a font by calling the functions `ATSFontActivateFromFileSpecification` (page 648) or `ATSFontActivateFromMemory` (page 649).

iRefCon

An arbitrary 32-bit value specified. This parameter is currently reserved for future use, so you should pass `NULL`.

iOptions

An `ATSOptionFlags` value. You should pass `kATSOptionFlagsDefault` unless to plan to call this function a number of times to deactivate many fonts. If you plan to call this function a number of times, you can set the `iOptions` parameter to `kATSOptionFlagsDoNotNotify` set. When you are done deactivating fonts you can call the function `ATSFontNotify` (page 676) with the `action` parameter set to `kATSFontNotifyActionFontsChanged`. ATS notifies all applications who subscribe to notifications of the changes you made.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
When you deactivate a font, you must supply the font container reference you obtained when you activated the font. You can't deactivate a font that you did not activate by calling the functions `ATSFontActivateFromFileSpecification` (page 648) or `ATSFontActivateFromMemory` (page 649).

You should use caution if you deactivate a font that is available globally, as its deactivation impacts any application that uses that font.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyApplyFunction

Applies your callback to a font family iteration.

```
OSStatus ATSFontFamilyApplyFunction (
   ATSFontFamilyApplierFunction iFunction,
   void *iRefCon
);
```

Parameters

`iFunction`
The callback function you want applied to a font family iteration. See `ATSFontApplierFunction` (page 682) for more information on the callback you need to supply.

`iRefCon`
An arbitrary 32-bit value specified by your application. This value is passed to your callback.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
The function `ATSFontFamilyApplyFunction` iterates through the default font families, which include globally activated font families and font families activated locally to your application. Calling this function is similar to creating an iterator that operates on a local context with an unrestricted scope.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyFindFromName

Returns the font family reference associated with a font family name.

```
ATSFontFamilyRef ATSFontFamilyFindFromName (
   CFStringRef iName,
   ATSOptionFlags iOptions
);
```

Parameters

`iName`
A reference to a font family name, formatted as a `CFString`.

iOptions
An `ATSOptionFlags` value. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

Return Value
A reference to the font family specified by the `iName` parameter. See the description of the `ATSFontFamilyRef` data type.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyFindFromQuickDrawName

Returns the font family reference associated with a standard QuickDraw font name.

```
ATSFontFamilyRef ATSFontFamilyFindFromQuickDrawName (
   ConstStr255Param iName
);
```

Parameters

iName
A QuickDraw font name.

Return Value
A reference to the font family associated with the font name specified by the `iName` parameter. See the description of the `ATSFontFamilyRef` data type.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyGetEncoding

Returns the text encoding used by a font family.

```
TextEncoding ATSFontFamilyGetEncoding (
   ATSFontFamilyRef iFamily
);
```

Parameters

iFamily
A font family reference.

Return Value
On output, a pointer to the text encoding used by the font family associated with the font family reference. See the Text Encoding Conversion Manager documentation for a description of the `TextEncoding` data type.

Discussion
Once you have obtained the text encoding, you can use Text Encoding Converter Manager function `RevertTextEncodingToScriptInfo` to extract the script as follows:

```
status = ATSFontFamilyGetEncoding (myFontFamily, &myTextEncoding)

status = RevertTextEncodingToScriptInfo (myTextEncoding, &myScriptCode);
```

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyGetGeneration

Returns the generation count of a font family.

```
ATSGeneration ATSFontFamilyGetGeneration (
   ATSFontFamilyRef iFamily
);
```

Parameters

iFamily
A font family reference.

Return Value
On output, the generation count for the font family associated with the font family reference. See the description of the `ATSGeneration` data type.

Discussion
The generation of a font family changes any time part of a font family is removed or added.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyGetName

Obtains the font family name associated with a font family reference.

```
OSStatus ATSFontFamilyGetName (
   ATSFontFamilyRef iFamily,
   ATSOptionFlags iOptions,
   CFStringRef *oName
);
```

Parameters

iFamily
A font family reference.

iOptions

An `ATSOptionFlags` value. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

oName

On output, a reference to the name associated with the font family reference, formatted as a `CFString`. You are responsible for releasing the `CFStringRef`.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyGetQuickDrawName

Obtains the standard QuickDraw font name associated with a font family reference.

```
OSStatus ATSFontFamilyGetQuickDrawName (
   ATSFontFamilyRef iFamily,
   Str255 oName
);
```

Parameters

iName

A reference to the font family name whose QuickDraw name you want to obtain.

oName

On input, a `Str255` value allocated by your application. On output, the QuickDraw name associated with the font family reference.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
All font families are assigned a QuickDraw name by the system. The QuickDraw name is almost identical to the font family name.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyIteratorCreate

Creates a font family iterator that your application can use to access font family objects.

```
OSStatus ATSFontFamilyIteratorCreate (
   ATSFontContext iContext,
   const ATSFontFilter *iFilter,
   void *iRefCon,
   ATSOptionFlags iOptions,
   ATSFontFamilyIterator *ioIterator
);
```

Parameters

iContext

A value that specifies the context of the iterator. If you want to apply the font family iterator only to the fonts accessible from your application use the `kATSFontContextLocal` constant. If you want the to apply the font family iterator to all fonts registered with the system use the constant `kATSFontContextGlobal`. See "Context Options" (page 703) for more information on the constants you can supply. See the Discussion for information on the interaction between the `iContext` and `iOptions` parameters.

iFilter

A pointer to a filter specification. Pass `NULL` if you do not want to apply a filter to this iteration. Otherwise, you can use this parameter to restrict the iteration to the font families that match a generation count or criteria you specify in a custom filter function. Pass the filter selector constant `kATSFontFilterSelectorGeneration` to select a generation filter or the constant `kATSFontFilterSelectorFontApplierFunction` to select a custom filter. See "Font Filter Selectors" (page 705) for more information on the constants you can supply.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. Otherwise, pass `NULL`.

iOptions

A value that specifies the scope of the iterator. If you want to iterate through font families that can be used only by your application, pass the constant `kATSOptionFlagsRestrictedScope`. If you want to iterate through font families that can be used by all applications pass the constant `kATSOptionFlagsUnRestrictedScope`. See "Scoping Options" (page 710) for more information on the constants you can supply. See the Discussion for information on the interaction between the `iContext` and `iOptions` parameters.

ioIterator

A pointer to a font family iterator. On output, points to an opaque font family iterator ready for you to use. When you no longer need the font family iterator, you should call the function `ATSFontFamilyIteratorRelease` (page 658) to release the auxiliary data and memory allocated by the system.

Return Value

A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion

Your application can use a font family iterator to access font family objects. A font family iterator is an opaque data structure used by ATS for Fonts to keep track of an iteration over currently active font families. When the font family iterator is initialized, it does not yet reference a font family.

The context and scope you specify for the font family iterator interact as shown in Table 31-1 (page 657).

Table 31-1 The interaction of context and scope in a font family enumeration

	Local context	Global context
Restricted scope	Font families activated locally to your application	Only globally activated font families
Unrestricted scope	Defaults font families, which include globally activated font families and font families activated locally to your application	All font families, which include globally activated font families and all other font families activated locally for an application.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyIteratorNext

Obtains the next font family reference.

```
OSStatus ATSFontFamilyIteratorNext (
   ATSFontFamilyIterator iIterator,
   ATSFontFamilyRef *oFamily
);
```

Parameters

iIterator
A pointer to a font family iterator you created with the function `ATSFontFamilyIteratorCreate` (page 655).

oFamily
A pointer to a font family reference. On output, points to the font family reference obtained by the iterator. You are responsible for allocating memory for the font family reference.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
If any changes are made to the font database while you are using the font family iterator, the iterator is invalidated and the function `ATSFontFamilyIteratorNext` returns the error `kATSIterationScopeModified`. To remedy this error, your application must either restart or cancel the enumeration by calling the `ATSFontFamilyIteratorReset` (page 658) or the `ATSFontFamilyIteratorRelease` (page 658) functions.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyIteratorRelease

Releases the memory associated with a font family iterator.

```
OSStatus ATSFontFamilyIteratorRelease (
   ATSFontFamilyIterator *ioIterator
);
```

Parameters

ioIterator

A pointer to a font family iterator you created with the function `ATSFontFamilyIteratorCreate` (page 655). If you try to use the font family iterator after disposing of its contents through this function, ATS for Fonts returns an error code to your application.

Return Value

A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion

If you plan to use the font family iterator again, you should consider calling the function `ATSFontFamilyIteratorReset` rather than releasing the font family iterator and then creating it again.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X 10.0 and later.

Declared In

`ATSFont.h`

ATSFontFamilyIteratorReset

Resets a font family iterator to the beginning of the iteration.

```
OSStatus ATSFontFamilyIteratorReset (
   ATSFontContext iContext,
   const ATSFontFilter *iFilter,
   void *iRefCon,
   ATSOptionFlags iOptions,
   ATSFontFamilyIterator *ioIterator
);
```

Parameters

iContext

A value that specifies the context of the iterator. If you want to apply the font family iterator only to the fonts accessible from your application use the `kATSFontContextLocal` constant. If you want the to apply the font family iterator to all fonts registered with the system use the constant `kATSFontContextGlobal`. See "Context Options" (page 703) for more information.

iFilter

A pointer to a filter specification. Pass `NULL` if you do not want to apply a filter to this iteration. Otherwise, you can use this parameter to restrict the iteration to the font families that match a generation count or criteria you specify in a custom filter function. Pass the filter selector constant `kATSFontFilterSelectorGeneration` to select a generation filter or the constant `kATSFontFilterSelectorFontApplierFunction` to select a custom filter. See "Font Filter Selectors" (page 705) for more information on these constants.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. If you are not using a custom filter function, pass `NULL`.

iOptions

An value that specifies the scope of the iterator. If you want to iterate through font families that can be used only by your application, pass the constant `kATSOptionFlagsRestrictedScope`. If you want to iterate through font families that can be used by all applications pass the constant `kATSOptionFlagsUnRestrictedScope`.

ioIterator

A pointer to a font family iterator you created with the function `ATSFontFamilyIteratorCreate` (page 655). On output, the font family iterator is reset to the beginning of the iteration.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
Once you have created a font family iterator, you can reuse it by calling the function `ATSFontFamilyIteratorReset`. This function sets the parameters to the new values you specify, and repositions the iterator so it is ready to get the first font family reference when you call the function `ATSFontFamilyIteratorNext` (page 657).

During an iteration, if you obtain the result code `kATSIterationScopeModified` from the function `ATSFontFamilyIteratorNext` (page 657), you can reset the iteration by calling the function `ATSFontFamilyIteratorReset`. This assures that you obtain the most up-to-date information from the iteration.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFindFromContainer

Obtains the font references contained in a font container.

```
OSStatus ATSFontFindFromContainer (
   ATSFontContainerRef iContainer,
   ATSOptionFlags iOptions,
   ItemCount iCount,
   ATSFontRef ioArray[],
   ItemCount *oCount
);
```

Parameters

iContainer

A reference to the font container whose fonts you want to obtain. You obtain a font container reference when you activate a font by calling the functions `ATSFontActivateFromFileSpecification` (page 648) or `ATSFontActivateFromMemory` (page 649).

iOptions
An ATSOptionFlags value. This parameter is currently reserved for future use, so you should pass kATSOptionFlagsDefault.

iCount
The number of items in the ioArray array. If you are uncertain of how many items are in this array, see the Discussion.

ioArray
A pointer to memory you have allocated for an array of font references. On return, the array contains the font references in the font container specified by the iContainer parameter. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oCount
A pointer to an ItemCount value. On output, the value specifies the actual number of ATSFontRef values in the font container.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
The function ATSFontFindFromContainer operates on font containers that reference font files. It does not work on font containers that reference font directories.

Typically you use the function ATSFindFontFromContainer by calling it twice, as follows:

1. Pass a reference to the font container to examine in the iContainer parameter, a valid pointer to an ItemCount value in the *oCount* parameter, NULL for the ioArray parameter, and 0 for the *iCount* parameter. ATSFindFontFromContainer returns the size of the array in the oCount parameter.

2. Allocate enough space for an array of the returned size, then call the ATSFindFontFromContainer function again, passing a valid pointer in the ioArray parameter and the number of items in the array in the *iCount* parameter. On return, the pointer refers to an array of the font references contained in the font container.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
ATSFont.h

ATSFontFindFromName

Returns the font reference associated with a font name.

```
ATSFontRef ATSFontFindFromName (
   CFStringRef iName,
   ATSOptionFlags iOptions
);
```

Parameters

iName
A reference to a font name formatted as a CFString.

iOptions
An `ATSOptionFlags` value. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

Return Value
A reference to the font specified by the `iName` parameter. See the description of the `ATSFontRef` data type.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontFindFromPostScriptName

Returns the font reference associated with a PostScript font name.

```
ATSFontRef ATSFontFindFromPostScriptName (
   CFStringRef iName,
   ATSOptionFlags iOptions
);
```

Parameters
iName
A reference to the PostScript name for a font, formatted as a `CFString`.

iOptions
An `ATSOptionFlags` value. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

Return Value
A reference to the font specified by the `iName` parameter. See the description of the `ATSFontRef` data type.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetAutoActivationSettingForApplication

Gets the activation setting for the specified application.

```
ATSFontAutoActivationSetting ATSFontGetAutoActivationSettingForApplication (
   CFURLRef iApplicationFileURL
);
```

Parameters
iApplicationFileURL
A valid file URL for an application. Pass `NULL` to specify the current process.

Return Value
The activation setting for the specified application.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`ATSFont.h`

ATSFontGetContainer

Gets the font container reference for a font.

```
OSStatus ATSFontGetContainer (
   ATSFontRef iFont,
   ATSOptionFlags iOptions,
   ATSFontContainerRef *oContainer
);
```

Parameters

iFont
The font reference.

iOptions
An options flag.

oContainer
On output, a reference to the font container that was used to activate the font reference. On error ATS sets this to `kATSFontContainerRefUnspecified`.

Return Value
If successful, `noErr`; if the container is invalid, `kATSInvalidFontContainerAccess`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`ATSFont.h`

ATSFontGetContainerFromFileReference

Gets the font container reference associated with an activated file reference.

```
OSStatus ATSFontGetContainerFromFileReference (
   const FSRef *iFile,
   ATSFontContext iContext,
   ATSOptionFlags iOptions,
   ATSFontContainerRef *oContainer
);
```

Parameters

iFile
A pointer to the valid file reference that specifies the activated font file for which to get the container.

iContext

The context that the font file is accessible to. If you want the activated font to be accessible only from your application pass `kATSFontContextDefault` or `kATSFontContextLocal`. If you want the activated font to be accessible to all applications use the constant `kATSFontContextGlobal`. See "Context Options" (page 703) for more information.

iOptions

An options flag.

oContainer

On output, a reference to the font container representing the file reference activated in the specified context. On error or for a file that is not activated, ATS sets this to `kATSFontContainerRefUnspecified`.

Return Value
`noErr` or `paramErr` if one or more parameters are invalid.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`ATSFont.h`

ATSFontGetFileReference

Obtains the file reference for a font.

```
OSStatus ATSFontGetFileReference (
   ATSFontRef iFont,
   FSRef *oFile
);
```

Parameters

iFont

A reference to the font whose file reference you want to obtain.

oFile

On output, points to the file reference that specifies the name and location of a file or directory that contains the font data specified by the `iFont` parameter.

Return Value
If successful, `noErr`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`ATSFont.h`

ATSFontGetFileSpecification

Obtains the file specification for a font. (Deprecated in Mac OS X v10.5. Instead use `ATSFontGetFileReference` (page 663).)

```
OSStatus ATSFontGetFileSpecification (
   ATSFontRef iFont,
   ATSFSSpec *oFile
);
```

Parameters

iFont

A reference to the font whose file specification you want to obtain.

oFile

On output, points to the file specification that specifies the name and location of a file or directory that contains the font data specified by the `iFont` parameter.

Return Value

A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion

The function `ATSFontGetFileSpecification` obtains the file specification for a font, not the font container. You must call the functions `ATSFontActivateFromFileSpecification` (page 648) or `ATSFontActivateFromMemory` (page 649) to obtain a font container reference.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`ATSFont.h`

ATSFontGetFontFamilyResource

Obtains the font family resource for a font.

```
OSStatus ATSFontGetFontFamilyResource (
   ATSFontRef iFont,
   ByteCount iBufferSize,
   void *ioBuffer,
   ByteCount *oSize
);
```

Parameters

iFont

A font reference.

iBufferSize

The size of the buffer pointed to by the `ioBuffer` parameter. See the Discussion if you are unsure of the size of this buffer.

ioBuffer

On input, a pointer to memory you allocated for the font family resource. On output, points to the `FOND` resource for the font. Note that the `FOND` resource data is in big endian format, regardless of the native endian format of the Macintosh computer on which you make the function call. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oSize
On output, the actual size of the buffer.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
The function `ATSFontGetFontFamilyResource` provides a compatibility path for font families that use resources. Beginning with Mac OS X version 10.2, ATS for Fonts synthesizes `FOND` resources for OpenType fonts.

Typically you use the function `ATSFontGetFontFamilyResource` by calling it twice, as follows:

1. Pass a reference to the font to examine in the `iFont` parameter, a valid pointer in the *oSize* parameter, `NULL` for the `ioBuffer` parameter, and 0 for the *iBufferSize* parameter. `ATSFontGetFontFamilyResource` returns the size of the buffer in the `oSize` parameter.

2. Allocate enough space for an array of the returned size, then call the `ATSFontGetFontFamilyResource` function again, passing a valid pointer in the `ioBuffer` parameter, the size of the buffer in the *iBufferSize* parameter, and the appropriate values in the other parameters. On return, the pointer refers to an array of the font references contained in the font container.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetGeneration

Returns the generation count for a font.

```
ATSGeneration ATSFontGetGeneration (
   ATSFontRef iFont
);
```

Parameters
iFont
A font reference.

Return Value
A generation count. See the description of the `ATSGeneration` data type.

Discussion
ATS for Fonts increments the generation count for any changes to a font, including when the system synthesizes data for the font.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetGlobalAutoActivationSetting

Gets the user's global auto-activation setting.

```
ATSFontAutoActivationSetting ATSFontGetGlobalAutoActivationSetting (
   void
);
```

Return Value
The user's global auto-activation setting.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`ATSFont.h`

ATSFontGetHorizontalMetrics

Obtains the horizontal metrics for a font.

```
OSStatus ATSFontGetHorizontalMetrics (
   ATSFontRef iFont,
   ATSOptionFlags iOptions,
   ATSFontMetrics *oMetrics
);
```

Parameters

iFont

A reference to the font whose horizontal metrics you want to obtain.

iOptions

An options flag. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

oMetrics

On input, a valid pointer to an `ATSFontMetrics` (page 689) data structure. On output, the structure contains the font's horizontal metrics. If one or more measurements are not available for a font, then the appropriate fields in the `ATSFontMetrics` data structure are set to `0`.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetName

Obtains the name of a font associated with a font reference.

```
OSStatus ATSFontGetName (
   ATSFontRef iFont,
   ATSOptionFlags iOptions,
   CFStringRef *oName
);
```

Parameters

iFont

A font reference.

iOptions

An `ATSOptionFlags` value. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

oName

On output, a reference to the font name associated with the specified font reference, formatted as a `CFString`. You are responsible for releasing the `CFStringRef`.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetPostScriptName

Obtains the PostScript name from a font reference.

```
OSStatus ATSFontGetPostScriptName (
   ATSFontRef iFont,
   ATSOptionFlags iOptions,
   CFStringRef *oName
);
```

Parameters

iFont

A font reference.

iOptions

An `ATSOptionFlags` value. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

oName

On output, a reference to the PostScript name for the font, formatted as a `CFString`. You are responsible for releasing the `CFStringRef`.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
The system automatically detects whether or not the font is composed PostScript. If the font is, ATS for Fonts appends the CMAP name.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetTable

Obtains a font table.

```
OSStatus ATSFontGetTable (
   ATSFontRef iFont,
   FourCharCode iTag,
   ByteOffset iOffset,
   ByteCount iBufferSize,
   void *ioBuffer,
   ByteCount *oSize
);
```

Parameters

iFont
A reference to the font whose table you want to obtain.

iTag
A four-character code that specifies the font table you want to obtain.

iOffset
The offset to a font table. If you want to obtain all the font tables associated with a font, pass `0`.

iBufferSize
The size of the buffer pointed to by the `ioBuffer` parameter. The size should be the actual size of the buffer (`oSize`) minus the offset to the font table (`iOffset`) you want to obtain. See the Discussion if you are unsure of value to supply.

ioBuffer
On input, a valid pointer. On output, a pointer to the font table. Note that the data returned in the font table is in big endian format, regardless of the native endian format of the Macintosh computer on which you make the function call. See the Discussion for information on allocating this buffer.

oSize
On output, the actual size of the buffer returned in the `ioBuffer` parameter.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
Synthetic font tables (entries with offset of 0) can only be accessed by calling the function `ATSFontGetTable`.

Typically you use the function `ATSFontGetTable` by calling it twice, as follows:

1. Pass a reference to the font whose table you want obtain in the `iFont` parameter, a four-character code that specifies the font table you want to obtain in the `iTag` parameter, the appropriate offset to the font table in the *`iOffset`* parameter, `0` for the *`iBufferSize`* parameter, `NULL` for the `ioBuffer` parameter, and a valid pointer to a `ByteCount` value in the *`oSize`* parameter. `ATSUFontGetTable` returns the size of the table in the `oSize` parameter.

2. Allocate enough space for a buffer of the returned size, then call the `ATSFontGetTable` function again, passing a valid pointer in the `ioBuffer` parameter, the size of the buffer in the *`iBufferSize`* parameter, and the appropriate values in the other parameters. On return, the pointer refers to the table for the font specified by the `iFont` parameter and the table specified by the `iTag` parameter.

You should use the function `ATSFontGetTable` when you need to obtain an entire font table. For performance reasons, avoid using the function to check a single value in the table.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetTableDirectory

Obtains the table directory for a font.

```
OSStatus ATSFontGetTableDirectory (
   ATSFontRef iFont,
   ByteCount iBufferSize,
   void *ioBuffer,
   ByteCount *oSize
);
```

Parameters

`iFont`
The font reference whose table directory you want to obtain.

`iBufferSize`
The size of the buffer pointed to by the `ioBuffer` parameter. See the Discussion if you are unsure of the size of this buffer.

`ioBuffer`
On input, a valid pointer. On output, points to the table directory for the font specified by the `iFont` parameter. Note that the data returned in the table directory is in big endian format, regardless of the native endian format of the Macintosh computer on which you make the function call. See the Discussion for information on allocating this buffer.

`oSize`
On output, the actual size of the buffer returned in the `ioBuffer` parameter.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
If necessary, ATS for Fonts synthesizes font tables or data, replacing existing tables or data. ATS for Fonts synthesizes data on an as needed basis; if data is synthesized, the generation count of the font increases.

Typically you use the function `ATSFontGetTableDirectory` by calling it twice, as follows:

1. Pass a reference to the font whose table directory you want obtain in the `iFont` parameter, `0` for the *`iBufferSize`* parameter, `NULL` for the `ioBuffer` parameter, and a valid pointer to a `ByteCount` value in the *`oSize`* parameter. `ATSFontGetTableDirectory` returns the size of the table directory in the `oSize` parameter.

2. Allocate enough space for a buffer of the returned size, then call the `ATSFontGetTableDirectory` function again, passing a valid pointer in the `ioBuffer` parameter, and the size of the buffer in the *`iBufferSize`* parameter. On return, the pointer refers to the table directory for the font specified by the `iFont` parameter.

If you want to obtain a font table, call the function `ATSFontGetTable` (page 668).

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontGetVerticalMetrics

Obtains the vertical metrics for a font.

```
OSStatus ATSFontGetVerticalMetrics (
   ATSFontRef iFont,
   ATSOptionFlags iOptions,
   ATSFontMetrics *oMetrics
);
```

Parameters

`iFont`
A reference to the font whose vertical metrics you want to obtain.

`iOptions`
An options flag. This parameter is currently reserved for future use, so you should pass `kATSOptionFlagsDefault`.

`oMetrics`
On input, a valid pointer to an `ATSFontMetrics` (page 689) data structure. On output, the structure contains the font's vertical metrics. If one or more measurements are not available for a font, then the appropriate fields in the `ATSFontMetrics` data structure are set to `0`.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontIsEnabled

Returns `true` if the font is enabled.

```
Boolean ATSFontIsEnabled (
   ATSFontRef iFont
);
```

Parameters

iFont

The font reference.

Return Value

`true` if the font is enabled.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`ATSFont.h`

ATSFontIteratorCreate

Creates a font iterator.

```
OSStatus ATSFontIteratorCreate (
   ATSFontContext iContext,
   const ATSFontFilter *iFilter,
   void *iRefCon,
   ATSOptionFlags iOptions,
   ATSFontIterator *ioIterator
);
```

Parameters

iContext

A value that specifies the context of the iterator. If you want to apply the font iterator only to the fonts accessible from your application use the `kATSFontContextLocal` constant. If you want the to apply the font iterator to all fonts registered with the system use the constant `kATSFontContextGlobal`. See "Context Options" (page 703) for more information on the constants you can supply. See the Discussion for information on the interaction between the `iContext` and `iOptions` parameters.

iFilter

A pointer to a filter specification. Pass `NULL` if you do not want to apply a filter to this iteration. Otherwise, you can use this parameter to restrict the iteration to the fonts that match a generation count or criteria you specify in a custom filter function. Pass the filter selector constant `kATSFontFilterSelectorGeneration` to select a generation filter or the constant `kATSFontFilterSelectorFontApplierFunction` to select a custom filter. See "Font Filter Selectors" (page 705) for more information on these constants.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. Otherwise, pass `NULL`.

iOptions

A value that specifies the scope of the iterator. If you want to iterate through fonts that can be used only by your application, pass the constant `kATSOptionFlagsRestrictedScope`. If you want to iterate through fonts that can be used by all applications pass the constant `kATSOptionFlagsUnRestrictedScope`. See "Scoping Options" (page 710) for more information on the constants you can supply. See the Discussion for information on the interaction between the `iContext` and `iOptions` parameters.

ioIterator

A pointer to a font iterator. On input, pass a pointer to an uninitialized iterator. On output, the iterator's contents may have been changed and may include references to data structures allocated by the system to maintain the iterator's state. When you no longer need the font iterator, you should call the function `ATSFontIteratorRelease` (page 673) to release the auxiliary data and memory allocated by the system.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
Your application can use a font iterator to access font data. A font iterator is an opaque data structure used by ATS for Fonts to keep track of an iteration over currently active fonts. When the font iterator is initialized, it does not yet reference a font.

The context and scope you specify for the font iterator interact as shown in Table 31-2 (page 672).

Table 31-2 The interaction of context and scope in a font enumeration

	Local context	Global context
Restricted scope	Fonts activated locally to your application	Only globally activated fonts
Unrestricted scope	Defaults fonts, which include globally activated fonts and fonts activated locally to your application	All fonts, which include globally activated fonts and all other fonts activated locally for an application.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontIteratorNext

Obtains the next font reference.

```
OSStatus ATSFontIteratorNext (
   ATSFontIterator iIterator,
   ATSFontRef *oFont
);
```

Parameters

iIterator

A pointer to a font iterator you created with the function `ATSFontIteratorCreate` (page 671). If you try to use the font iterator after disposing of its contents through this function, ATS for Fonts returns an error code to your application.

oFont

A pointer to a font reference. On output, points to the font reference obtained by the iterator. You are responsible for allocating memory for the font reference.

Return Value

A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion

If any changes are made to the font database while you are using the font iterator, the iterator is invalidated and the function `ATSFontFamilyIteratorNext` returns the error `kATSIterationScopeModified`. To remedy this error, your application must either restart or cancel the enumeration by calling the `ATSFontFamilyIteratorReset` (page 658) or the `ATSFontIteratorRelease` (page 673) functions.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X 10.0 and later.

Declared In

`ATSFont.h`

ATSFontIteratorRelease

Releases a font iterator.

```
OSStatus ATSFontIteratorRelease (
   ATSFontIterator *ioIterator
);
```

Parameters

ioIterator

A pointer to a font iterator you created with the function `ATSFontIteratorCreate` (page 671). If you try to use the font iterator after disposing of its contents through this function, ATS for Fonts returns an error code to your application.

Return Value

A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion

If you plan to use the font iterator again, you should consider calling the function `ATSFontIteratorReset` rather than releasing the font iterator and then creating it again.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X 10.0 and later.

Declared In
ATSFont.h

ATSFontIteratorReset

Resets a font iterator to the beginning of the iteration.

```
OSStatus ATSFontIteratorReset (
   ATSFontContext iContext,
   const ATSFontFilter *iFilter,
   void *iRefCon,
   ATSOptionFlags iOptions,
   ATSFontIterator *ioIterator
);
```

Parameters

iContext

A value that specifies the context of the iterator. If you want to apply the font iterator only to the fonts accessible from your application use the kATSFontContextLocal constant. If you want the to apply the font iterator to all fonts registered with the system use the constant kATSFontContextGlobal. See "Context Options" (page 703) for more information.

iFilter

A pointer to a filter specification. Pass NULL if you do not want to apply a filter to this iteration. Otherwise, you can use this parameter to restrict the iteration to the fonts that match a generation count or criteria you specify in a custom filter function. Pass the filter selector constant kATSFontFilterSelectorGeneration to select a generation filter or the constant kATSFontFilterSelectorFontApplierFunction to select a custom filter. See "Font Filter Selectors" (page 705) for more information on these constants.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. Otherwise, pass NULL.

iOptions

A value that specifies the scope of the iterator. If you want to iterate through fonts that can be used only by your application, pass the constant kATSOptionFlagsRestrictedScope. If you want to iterate through fonts that can be used by all applications pass the constant kATSOptionFlagsUnRestrictedScope.

ioIterator

A pointer to a font iterator you created with the function ATSFontIteratorCreate (page 671). If you try to use the font iterator after disposing of its contents through this function, ATS for Fonts returns an error code to your application.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
Once you have created a font iterator, you can reuse it by calling the function ATSFontIteratorReset. This function sets the parameters to the new values you specify, and repositions the iterator so it is ready to get the first font reference when you call the function ATSFontIteratorNext (page 672).

During an iteration, if you obtain the result code kATSIterationScopeModified from the function ATSFontIteratorNext (page 672), you can reset the iteration by calling the function ATSFontIteratorReset. This assures that you obtain the most up-to-date information from the iteration.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

ATSFontNotificationSubscribe

Signs up your application to receive notification of changes to fonts and font directories.

```
OSStatus ATSFontNotificationSubscribe (
   ATSNotificationCallback callback,
   ATSFontNotifyOption options,
   void *iRefcon,
   ATSFontNotificationRef *oNotificationRef
);
```

Parameters

callback

The callback function you want ATS to invoke whenever the notification action specified in the `options` parameter occurs. See `ATSNotificationCallback` (page 684) for more information on the callback you can supply.

options

A notification option that specifies when you want ATS to respond to notification actions. If you want to receive notifications when your application is in the foreground, pass the constant `kATSFontNotifyOptionDefault`. If your application is a server process or a tool that performs font management functions and requires immediate notification when fonts change, pass the constant `kATSFontNotifyOptionReceiveWhileSuspended`. See "Notification Options" (page 709) for more information.

iRefCon

An arbitrary 32-bit value specified by your application and which you want passed to your callback function. You can pass `NULL` if your callback does not need any data.

oNotificationRef

On output, a notification reference. You need this reference when you call the function `ATSFontNotificationUnsubscribe` (page 676). You can pass `NULL` if you do not want to obtain the reference.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715). This function returns `paramErr` the `callback` parameter is `NULL` and `memFullErr` if the function cannot allocate enough memory for internal data structures.

Discussion
If your application uses Carbon events or the Application Kit, you can call the function `ATSFontNotificationSubscribe` to receive notifications of changes to fonts and font directories. However, if your application is of a type that does not use a `CFRunLoop`, it can't receive notifications unless you explicitly set up a run loop. For more information on run loops, see *Overview of Programming Topic: Run Loops* on the Cocoa Developer Documentation website.

If you want to stop receiving notifications, call the function `ATSFontNotificationUnsubscribe` (page 676).

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.2 and later.

Declared In
`ATSFont.h`

ATSFontNotificationUnsubscribe

Unsubscribes your application from receiving notifications of changes to fonts and font directories.

```
OSStatus ATSFontNotificationUnsubscribe (
   ATSFontNotificationRef notificationRef
);
```

Parameters

oNotificationRef

On input, the notification reference you obtained when you called the function `ATSFontNotificationSubscribe` (page 675). On output, `NULL`.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715). Returns `paramErr` if you pass a `NULL` or invalid notification reference in the `oNotificationRef` parameter.

Discussion
The function `ATSFontNotificationUnsubscribe` unsubscribes your application from receiving the notification associated with the notification reference you pass to the function. You must call `ATSFontNotificationUnsubscribe` for each notification for which you want to unsubscribe.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.2 and later.

Declared In
`ATSFont.h`

ATSFontNotify

Notifies Apple Type Services of an action taken by your application.

```
OSStatus ATSFontNotify (
   ATSFontNotifyAction action,
   void *info
);
```

Parameters

action

A notification action that specifies the action taken by your application. If your application activates or deactivates fonts, you should pass `kATSFontNotifyActionFontsChanged`. If your application makes changes to any of the font directories (System, local, user, or the Classic System folder), you should pass the constant `kATSFontNotifyActionDirectoriesChanged`. See "Notification Actions" (page 708) for more information on the constants you can supply.

info

A pointer to the data you want ATS for Fonts to pass to the clients who subscribe to notifications. You can pass `NULL` if there is no data associated with this action.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715). Returns `paramErr` if you pass an invalid notification action in the `action` parameter.

Discussion
A notification is a mechanism by which your application can inform ATS for Fonts that you have changed a font or font directory. Other applications can sign up to receive notifications of these changes by calling the function `ATSFontNotificationSubscribe` (page 675). When you call the function `ATSFontNotify`, the system passes the notification along with any data you provide to every client who is signed up to receive notifications.

You can call the function `ATSFontNotify` after your application makes a batch of changes. For example, if your application calls the functions `ATSFontActivateFromFileSpecification` (page 648) or `ATSFontDeactivate` (page 651) multiple times to activate and deactivate fonts, you can set the `iOptions` parameter in these functions to `kATSOptionFlagsDoNotNotify` set. When you are done activating and deactivating fonts you can call the function `ATSFontNotify` with the `action` parameter set to `kATSFontNotifyActionFontsChanged`. Then ATS notifies all applications who subscribe to notifications of the changes you made.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.2 and later.

Declared In
`ATSFont.h`

ATSFontSetAutoActivationSettingForApplication

Sets the auto-activation setting for the specified application bundle.

```
OSStatus ATSFontSetAutoActivationSettingForApplication (
   ATSFontAutoActivationSetting iSetting,
   CFURLRef iApplicationFileURL
);
```

Parameters

iSetting

A font auto-activation setting. See "Automatic Activation Settings" (page 703).

iApplicationFileURL

A valid file URL for an application. Pass `NULL` to specify the current process.

Return Value
Returns `noErr` on success, and `paramErr` for any invalid input. May return `memFullErr` if unable to allocate temporary structures.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`ATSFont.h`

ATSFontSetEnabled

Sets a font state to enabled or disabled.

```
OSStatus ATSFontSetEnabled (
   ATSFontRef iFont,
   ATSOptionFlags iOptions,
   Boolean iEnabled
);
```

Parameters

iFont

The font reference.

iOptions

An options flag.

iEnabled

The state to set the font to. `True` for enabled, `false` for disabled.

Return Value

`kATSInvalidFontAccess` if the font reference is invalid in the current application context.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`ATSFont.h`

ATSFontSetGlobalAutoActivationSetting

Sets the user's global auto-activation setting.

```
OSStatus ATSFontSetGlobalAutoActivationSetting (
   ATSFontAutoActivationSetting iSetting
);
```

Parameters

iSetting

A font auto-activation setting. See "Automatic Activation Settings" (page 703).

Return Value

If successful, `noErr`; if invalid input, `paramErr`.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`ATSFont.h`

ATSGetGeneration

Obtains the generation of the font database.

```
ATSGeneration ATSGetGeneration (
   void
);
```

Return Value
A value that specifies the generation count of the font database. See the description of the `ATSGeneration` data type.

Discussion
Any operation that adds, deletes, or modifies one or more font families or fonts triggers an update of the font database generation count. If you want to obtain the generation of a font family, call the function `ATSFontFamilyGetGeneration` (page 654). If you want to obtain the generation of a font, call the function `ATSFontGetGeneration` (page 665).

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X 10.0 and later.

Declared In
`ATSFont.h`

DisposeFMFontCallbackFilterUPP

Disposes of a universal procedure pointer to a customized filter function used for fonts.

```
void DisposeFMFontCallbackFilterUPP (
   FMFontCallbackFilterUPP userUPP
);
```

Discussion
See the callback `FMFontCallbackFilterProcPtr` (page 685) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`ATSTypes.h`

DisposeFMFontFamilyCallbackFilterUPP

Disposes of a universal procedure pointer to a customized filter function used for font families.

```
void DisposeFMFontFamilyCallbackFilterUPP (
   FMFontFamilyCallbackFilterUPP userUPP
);
```

Discussion
See the callback `FMFontFamilyCallbackFilterProcPtr` (page 686) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
ATSTypes.h

InvokeFMFontCallbackFilterUPP

Calls a customized filter function used for fonts.

```
OSStatus InvokeFMFontCallbackFilterUPP (
   FMFont iFont,
   void *iRefCon,
   FMFontCallbackFilterUPP userUPP
);
```

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
This function is not recommended nor needed, as the system invokes your filter for you.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
ATSTypes.h

InvokeFMFontFamilyCallbackFilterUPP

Calls a customized filter function used for font families.

```
OSStatus InvokeFMFontFamilyCallbackFilterUPP (
   FMFontFamily iFontFamily,
   void *iRefCon,
   FMFontFamilyCallbackFilterUPP userUPP
);
```

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
This function is not recommended nor needed, as the system invokes your filter for you.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
ATSTypes.h

NewFMFontCallbackFilterUPP

Creates a new universal procedure pointer (UPP) to a filter callback function that uses your criteria for filtering fonts.

```
FMFontCallbackFilterUPP NewFMFontCallbackFilterUPP (
   FMFontCallbackFilterProcPtr userRoutine
);
```

Return Value
See the description of the `FMFontCallbackFilterUPP` data type.

Discussion
See the callback `FMFontCallbackFilterProcPtr` (page 685) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`ATSTypes.h`

NewFMFontFamilyCallbackFilterUPP

Creates a new universal procedure pointer (UPP) to a filter callback function that uses your criteria for filtering font families.

```
FMFontFamilyCallbackFilterUPP NewFMFontFamilyCallbackFilterUPP (
   FMFontFamilyCallbackFilterProcPtr userRoutine
);
```

Return Value
See the description of the `FMFontFamilyCallbackFilterUPP` data type.

Discussion
See the callback `FMFontFamilyCallbackFilterProcPtr` (page 686) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`ATSTypes.h`

Callbacks by Task

ATS Callbacks

The callbacks in this section are used by ATS for Fonts.

`ATSFontApplierFunction` (page 682)
Defines a pointer to a customized function to be applied to a font iteration.

`ATSFontFamilyApplierFunction` (page 683)
Defines a pointer to a customized function to be applied to a font family iteration.

`ATSFontQueryCallback` (page 683)
Defines a pointer to a customized function that handles font queries.

`ATSNotificationCallback` (page 684)
Defines a pointer to a customized function that handles notifications.

FM Callbacks

The callbacks in this section are used by the Font Manager.

`FMFontCallbackFilterProcPtr` (page 685)
Defines a pointer to a customized filter function to be used with a font iterator.

`FMFontFamilyCallbackFilterProcPtr` (page 686)
Defines a pointer to a customized filter function to be used with a font family iterator.

Callbacks

ATSFontApplierFunction

Defines a pointer to a customized function to be applied to a font iteration.

```
typedef OSStatus (*ATSFontApplierFunction) (
    ATSFontRef iFont,
    void * iRefCon
);
```

If you name your function `MyATSFontApplierFunction`, you would declare it like this:

```
OSStatus MyATSFontApplierFunction (
    ATSFontRef iFont,
    void * iRefCon
);
```

Parameters

iFont
A font reference. This is the font on which your callback operates.

iRefCon
An arbitrary 32-bit value specified by your application and that is passed to your callback.

Return Value

A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion

You provide a pointer to an `ATSFontApplierFunction` callback as a parameter to the function `ATSFontApplyFunction` (page 650). You can also provide a pointer to an `ATSFontApplierFunction` callback in the `ATSFontFilter` (page 688) data structure. This structure can be passed as a parameter to the functions `ATSFontIteratorCreate` (page 671) and `ATSFontIteratorReset` (page 674).

Availability

Available in Mac OS X v10.0 and later.

Declared In
ATSFont.h

ATSFontFamilyApplierFunction

Defines a pointer to a customized function to be applied to a font family iteration.

```
typedef OSStatus (*ATSFontFamilyApplierFunction) (
    ATSFontFamilyRef iFamily,
    void * iRefCon
);
```

If you name your function `MyATSFontFamilyApplierFunction`, you would declare it like this:

```
OSStatus MyATSFontFamilyApplierFunction (
    ATSFontFamilyRef iFamily,
    void * iRefCon
);
```

Parameters

iFamily
A font family reference. This is the font family on which your callback operates.

iRefCon
An arbitrary 32-bit value specified by your application and that is passed to your callback.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
You provide a pointer to an `ATSFontFamilyApplierFunction` callback as a parameter to the function `ATSFontFamilyApplyFunction` (page 652). You can also provide a pointer to an `ATSFontApplierFunction` callback in the `ATSFontFilter` (page 688) data structure. This structure can be passed as a parameter to the functions `ATSFontFamilyIteratorCreate` (page 655) and `ATSFontFamilyIteratorReset` (page 658).

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSFont.h

ATSFontQueryCallback

Defines a pointer to a customized function that handles font queries.

```
typedef CFPropertyListRef (*ATSFontQueryCallback) (
    ATSFontQueryMessageID msgid,
     CFPropertyListRef data
       void * iRefCon
);
```

If you name your function `MyATSFontQueryCallback`, you would declare it like this:

```
CFPropertyListRef MyATSFontQueryCallback (
    ATSFontQueryMessageID msgid,
     CFPropertyListRef data
       void * iRefCon
);
```

Parameters

msgid

An `ATSFontQueryMessageID` value that identifies the message type your application receives from ATS. See "Font Query Message ID" (page 708) for the constants you can supply.

data

A `CFPropertyListRef` that represents the font query. The content of the `CFPropertyList` is specific to the message type. The property list should contain data that specifies the font for which the query is sent.

iRefCon

An arbitrary 32-bit value specified by your application and that is passed to your callback.

Return Value

A `CFPropertyListRef` that represents your application's response to the query. The content of the `CFPropertyList` is specific to the message type, and it may be `NULL`.

Discussion

ATS for Fonts calls your customized function each time ATS receives a font query from another application. You provide a pointer to an `ATSFontQueryCallback` as a parameter to the function `ATSCreateFontQueryRunLoopSource` (page 646).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSFont.h`

ATSNotificationCallback

Defines a pointer to a customized function that handles notifications.

```
typedef void (*ATSNotificationCallback) (
   ATSFontNotificationInfoRef info,
      void * iRefCon
);
```

If you name your function `MyATSNotificationCallback`, you would declare it like this:

```
void MyATSNotificationCallback (
   ATSFontNotificationInfoRef info,
      void * iRefCon
);
```

Parameters

info

Reserved for future use. Currently, your callback is passed `NULL`.

iRefCon
An arbitrary 32-bit value specified by your application and that is passed to your callback.

Discussion
ATS for Fonts calls your customized function each time ATS receives a font notification from another application. You provide a pointer to an `ATSNotificationCallback` callback function as a parameter to the function `ATSFontNotificationSubscribe` (page 675).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSFont.h`

FMFontCallbackFilterProcPtr

Defines a pointer to a customized filter function to be used with a font iterator.

```
typedef OSStatus (*FMFontCallbackFilterProcPtr) (
    FMFont iFont,
    void * iRefCon
);
```

If you name your function `MyFMFontCallbackFilterProc`, you would declare it like this:

```
OSStatus MyFMFontCallbackFilterProcPtr (
    FMFont iFont,
    void * iRefCon
);
```

Parameters

iFont
A font reference. This is the font on which your callback operates.

iRefCon
A pointer to arbitrary data that defines your custom filter.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
The Font Manager calls your customized function each time it obtains a font in a font iteration. You can use a custom filter function in any Font Manager function that has a parameter of type `FMFilter`. You provide a universal procedure pointer to your filter callback function in the `FMFilter` data structure. First, you must use the `NewFMFontCallbackFilterUPP` (page 680) function to create a universal procedure pointer (UPP) of type `FMFontCallbackFilterUPP`. You can do so with code similar to the following:

```
FMFontCallbackFilterUPP MyFMFontFilterUPPMyFMFontFilterUPP =
NewFMFontCallbackFilterUPP (&MyFMFontCallbackFilterCallback)
```

Your application must specify the result code that should be returned by the Font Manager. Any value other than `noErr` will cause the iterator to ignore a font.

When you are finished with your filter callback function, you should use the `DisposeFMFontCallbackFilterUPP` (page 679) function to dispose of the UPP associated with it. However, if you plan to use the same filter callback function in subsequent calls, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontFamilyCallbackFilterProcPtr

Defines a pointer to a customized filter function to be used with a font family iterator.

```
typedef OSStatus (*FMFontFamilyCallbackFilterProcPtr) (
    FMFontFamily iFontFamily,
    void * iRefCon
);
```

If you name your function `MyFMFontFamilyCallbackFilterProc`, you would declare it like this:

```
OSStatus MyFMFontFamilyCallbackFilterProcPtr (
    FMFontFamily iFontFamily,
    void * iRefCon
);
```

Parameters

iFontFamily
A font family reference. This is the font family on which your callback operates.

iRefCon
A pointer to arbitrary data that defines your custom filter.

Return Value
A result code. See "Apple Type Services for Fonts Result Codes" (page 715).

Discussion
The Font Manager calls your customized function each time it obtains a font family in a font family iteration. You can use a custom filter function in any Font Manager function that has a parameter of type `FMFilter`. You provide a universal procedure pointer to your filter callback function in the `FMFilter` data structure. First, you must use the function `NewFMFontFamilyCallbackFilterUPP` (page 681) to create a universal procedure pointer (UPP) of type `FMFontFamilyCallbackFilterUPP`. You can do so with code similar to the following:

```
FMFontFamilyCallbackFilterUPP MyFMFontFamilyFilterUPPMyFMFontFamilyFilterUPP =
NewFMFontFamilyCallbackFilterUPP (&MyFMFontFamilyCallbackFilterCallback)
```

Your application must specify the result code that should be returned by the Font Manager. Any value other than `noErr` will cause the iterator to ignore a font family.

When you are finished with your filter callback function, you should use the `DisposeFMFontFamilyCallbackFilterUPP` (page 679) function to dispose of the UPP associated with it. However, if you plan to use the same filter callback function in subsequent calls, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

Data Types

ATS Data Types

The data types in this section are used by ATS for Fonts.

ATSFontContainerRef

An opaque data type that represents a reference to a font file or folder.

```
typedef UInt32 ATSFontContainerRef;
```

Discussion
A font container reference is an opaque type used as a parameter in the functions `ATSFontActivateFromFileSpecification` (page 648) and `ATSFontActivateFromMemory` (page 649).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSFontFamilyIterator

An opaque data type that represents a font family iterator.

```
typedef struct ATSFontFamilyIterator_ * ATSFontFamilyIterator;
```

Discussion
You initialize a structure of type `ATSFontFamilyIterator` by calling the function `ATSFontFamilyIteratorCreate` (page 655). You should not attempt to modify the contents of a font family iterator.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSFont.h`

ATSFontFamilyRef

An opaque data type that represents a font family reference.

```
typedef UInt32 ATSFontFamilyRef;
```

Discussion
Unlike font family and font names which are part of a font's data, data types, such as `ATSFontFamily` represent values that are arbitrarily assigned by ATS at system startup. These values can change when the system is restarted.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSFontFilter

Contains font filter information.

```
struct ATSFontFilter {
    UInt32 version
    ATSFontFilterSelector filterSelector
    union {
        ATSGeneration generationFilter;
        ATSFontFamilyRef fontFamilyFilter;
        ATSFontFamilyApplierFunction fontFamilyApplierFunctionFilter;
        ATSFontApplierFunction fontApplierFunctionFilter;
    } filter;
};
typedef struct ATSFontFilter ATSFontFilter;
```

Fields

`version`
The version of the filter.

`filterSelector`
A font filter selector. See "Font Filter Selectors" (page 705)for a list of the filter selectors you can specify.

`filter`
A union whose contents are specified by the `filterSelector` field.

`generationFilter`
An `ATSGeneration` value that specifies the generation to which you want to restrict an operation.

`fontFamilyFilter`
A font family reference that specifies the font family to which you want to restrict an operation.

`fontFamilyApplierFunctionFilter`
A pointer the callback you want applied to a font family iteration. See `ATSFontFamilyApplierFunction` (page 683) for more information on the callback you can supply.

`fontApplierFunctionFilter`
A pointer the callback you want applied to a font iteration. See `ATSFontApplierFunction` (page 682) for more information on the callback you can supply.

Discussion
You can pass an `ATSFontFilter` structure to the functions `ATSFontFamilyIteratorCreate` (page 655), `ATSFontFamilyIteratorReset` (page 658), `ATSFontIteratorCreate` (page 671), and `ATSFontIteratorReset` (page 674).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSFont.h`

ATSFontIterator

An opaque data type that represents a font iterator.

```
typedef struct ATSFontIterator_ * ATSFontIterator;
```

Discussion
You initialize a structure of type `ATSFontIterator` by calling the function `ATSFontIteratorCreate` (page 671). You should not attempt to modify the contents of a font iterator.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSFont.h`

ATSFontMetrics

Contains metrics for a font.

```
struct ATSFontMetrics {
    UInt32 version;
    Float32 ascent;
    Float32 descent;
    Float32 leading;
    Float32 avgAdvanceWidth;
    Float32 maxAdvanceWidth;
    Float32 minLeftSideBearing;
    Float32 minRightSideBearing;
    Float32 stemWidth;
    Float32 stemHeight;
    Float32 capHeight;
    Float32 xHeight;
    Float32 italicAngle;
    Float32 underlinePosition;
    Float32 underlineThickness;
};
typedef struct ATSFontMetrics ATSFontMetrics;
```

Fields

`version`
The version of the font metrics structure.

`ascent`
The maximum height from the baseline to the ascent line of the glyphs in the font. For vertical text, the maximum distance from the center line to the ascent line of the glyphs in the font.

`descent`
: The maximum distance from the baseline to the descent line of the the glyphs in the font. For vertical text, the maximum distance from center line to the descent line of the glyphs in the font.

`leading`
: The spacing from the descent line to the ascent line below it. This defines the spacing between lines of text

`avgAdvanceWidth`
: The average advance width of the glyph in the font.

`maxAdvanceWidth`
: The maximum advance width of the glyphs in the font.

`minLeftSideBearing`
: The minimum left-side bearing value of the glyphs in the font. For vertical text, the minimum top-side bearing value of the glyphs in the font.

`minRightSideBearing`
: The minimum right-side bearing value of the glyphs in the font. For vertical text, the minimum bottom side bearing of a glyphs in the font.

`stemWidth`
: The width of the dominant vertical stems of the glyphs in the font.

`stemHeight`
: The vertical width of the dominant horizontal stems of glyphs in the font.

`capHeight`
: The height of a capital letter in the font from the baseline to the top of the letter.

`xHeight`
: The height of lowercase characters in the font, specifically the letter x, excluding ascenders and descenders.

`italicAngle`
: The angle (in degrees counterclockwise) at which glyphs in the font slant when italicized.

`underlinePosition`
: The position at which an underline stroke should be placed for the font.

`underlineThickness`
: The thickness, in pixels, of the underscore character used to underline the glyphs in the font.

Discussion
This structure is passed as a parameter to the functions `ATSFontGetHorizontalMetrics` (page 666) and `ATSFontGetVerticalMetrics` (page 670).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSFontNotificationInfoRef

An opaque data type that represents a font notification information structure.

```
typedef struct ATSFontNotificationInfoRef_ * ATSFontNotificationInfoRef;
```

Discussion
This data type is used in the `ATSNotificationCallback` (page 684) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSFont.h`

ATSFontNotificationRef

An opaque data type that represents a font notification structure.

```
typedef struct ATSFontNotificationRef_ * ATSFontNotificationRef;
```

Discussion
The `ATSFontNotificationRef` data type is returned by the function `ATSFontNotificationSubscribe` (page 675) and passed as a parameter to the function `ATSFontNotificationUnsubscribe` (page 676).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSFont.h`

ATSFontQuerySourceContext

Contains font query information that is passed back to a font query callback.

```
struct ATSFontQuerySourceContext {
    UInt32 version;
    void * refCon;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
};
typedef struct ATSFontQuerySourceContext ATSFontQuerySourceContext;
```

Fields

`version`
A 32-bit unsigned integer that indicates the version of this data structure. You can set this value to `0`.

`refCon`
An arbitrary 32-bit value specified in your font query callback function.

`retain`
A callback you supply for increasing the retention count associated with the `refCon` value. The `CFAllocatorRetainCallBack` is defined in the header file `CFBase.h`. For more information on Core Foundation allocators, see the *Core Foundation Base Services Reference*.

`release`
A callback you supply for decreasing the retention count associated with the `refCon` value. The `CFAllocatorReleaseCallBack` is defined in the header file `CFBase.h`.

Discussion
You pass a `ATSFontQuerySourceContext` data structure to the function `ATSCreateFontQueryRunLoopSource` (page 646).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSFont.h`

ATSFontRef

An opaque data type that represents a font reference.

```
typedef UInt32 ATSFontRef;
```

Discussion
Unlike font names which are part of a font's data, data types, such as `ATSFontRef` represent values that are arbitrarily assigned by ATS at system startup. These values can change when the system is restarted.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSFontSize

Represents a font size.

```
typedef Float32 ATSFontSize;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSGeneration

Represents a generation count.

```
typedef UInt32 ATSGeneration;
```

Discussion
The generation count data type is used by ATS for Fonts to keep track of the generation of the font database, each font family, and each font. You can obtain a generation count from the functions `ATSGetGeneration` (page 678), `ATSFontFamilyGetGeneration` (page 654), and `ATSFontGetGeneration` (page 665).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSOptionFlags

Represents options you can pass to various ATS functions.

```
typedef OptionBits ATSOptionFlags;
```

Discussion
There are a variety of options associated with this data type. See "Assorted Options" (page 702), "Scoping Options" (page 710), and "Iteration Precedence Options" (page 708).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FM Data types

The data types in this section are used by the Font Manager.

FMFilter

Contains a filter format, a selector and filter information.

```
struct FMFilter {
    UInt32 format
    FMFilterSelector selector
    union {
        FourCharCode fontTechnologyFilter;
        FSSpec fontContainerFilter;
        FMGeneration generationFilter;
        FMFontFamilyCallbackFilterUPP fontFamilyCallbackFilter;
        FMFontCallbackFilterUPP fontCallbackFilter;
        FMFontDirectoryFilter fontDirectoryFilter;
    } filter;
};
typedef struct FMFilter FMFilter;
```

Fields

`format`
A filter format. For possible values, see FM Filter Format (page 711).

`selector`
A filter selector. The selector indicates the data contained in the union. For possible values, see "FM Filter Selectors" (page 711).

`filter`
: The filter you want to use to restrict an operation. The filter must correspond to the `selector` parameter. If you are using a custom filter, you should provide a universal procedure pointer that is either of type `FMFontFamilyCallbackFilterUPP` or `FMFontCallBackFilterUPP`.

`fontTechnologyFilter`
: A `FourCharCode` value that specifies the font technology to which you want to restrict an operation. See "FM Font Technologies" (page 712) for constants you can supply.

`fontContainerFilter`
: A pointer to the file specification that specifies the name and location of a file or directory to which you want to restrict an operation.

`generationFilter`
: The generation count to which you want to restrict an operation.

`fontFamilyCallbackFilter`
: The font family callback that you want to use to restrict an operation.

`fontCallbackFilter`
: The font callback that you want to use to restrict an operation.

`fontDirectoryFilter`
: The font directory filter that you want to use to restrict an operation.

Discussion
You use the `FMFilter` (page 693) data structure when you want to restrict the enumeration and activation functions to the criteria specified by a filter.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFont

An opaque data type that specifies a font registered with the font database.

```
typedef UInt32 FMFont;
```

Discussion
You should not modify this value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontCallbackFilterUPP

Defines a universal procedure pointer to a font filter callback.

```
typedef FMFontCallbackFilterProcPtr FMFontCallbackFilter;
```

Discussion
For more information, see the description of the `FMFontCallbackFilterProcPtr` (page 685) callback function.

FMFontDirectoryFilter

Contains font directory information used to restrict a font iteration.

```
struct FMFontDirectoryFilter {
    SInt16 fontFolderDomain;
    UInt32 reserved[2];
};
typedef struct FMFontDirectoryFilter FMFontDirectoryFilter;
```

Fields
`fontFolderDomain`
A signed 16-bit integer that specifies the directory to which you want to restrict the font iteration.
`reserved`
Reserved for future use.

Discussion
You supply the `FMFontDirectoryFilter` data structure as part of the `FMFilter` (page 693) data structure when you want to restrict a font iteration to a font directory.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontFamily

A reference to a collection of fonts with the same design characteristics.

```
typedef SInt16 FMFontFamily;
```

Discussion
The font family reference replaces the QuickDraw font ID and can be used with all QuickDraw functions including `GetFontName` and `TextFont`. Unlike the QuickDraw font identifier, the font family reference cannot be passed to the Resource Manager to access information from a `'FOND'` resource. A font family reference does not imply a script system, nor is the character encoding of a font family determined by an arithmetic mapping of the font family reference.

The fonts associated with a font family consist of individual outline fonts that may be used with the font access functions of the Font Manager and ATSUI.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontFamilyCallbackFilterUPP

Defines a universal procedure pointer to a font family filter callback.

```
typedef FMFontFamilyCallbackFilterProcPtr FMFontFamilyCallbackFilter;
```

Discussion
For more information, see the description of the `FMFontFamilyCallbackFilterProcPtr` (page 686) callback function.

FMFontFamilyInstance

Contains a font family reference and a QuickDraw style.

```
struct FMFontFamilyInstance {
    FMFontFamily fontFamily;
    FMFontStyle fontStyle;
};
typedef struct FMFontFamilyInstance FMFontFamilyInstance;
```

Fields
`fontFamily`
A font family reference.
`fontStyle`
A QuickDraw font style.

Discussion
Each font object can map to one or more font family instance. This mapping is equivalent to the information stored in the font association table of the `'FOND'` resource, except the font family instance does not contain a point size descriptor. Since a font object represents the entire array of point sizes for a given font, only the font family reference and style are required to specify fully any given font object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontFamilyInstanceIterator

An opaque structure used to enumerate font family instances.

```
struct FMFontFamilyInstanceIterator {
    UInt32 reserved[16];
};
typedef struct FMFontFamilyInstanceIterator FMFontFamilyInstanceIterator;
```

Fields
`reserved`
Reserved for Apple's use.

Discussion
You initialize a structure of type `FMFontFamilyInstanceIterator` by calling the function `FMCreateFontFamilyInstanceIterator`. You should not attempt to modify the contents of a font family instance iterator.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontFamilyIterator

An opaque structure used to enumerate font families.

```
struct FMFontFamilyIterator {
    UInt32 reserved[16];
};
typedef struct FMFontFamilyIterator FMFontFamilyIterator;
```

Fields
`reserved`
Reserved for Apple's use.

Discussion
You initialize a structure of type `FMFontFamilyIterator` by calling the function `FMCreateFontFamilyIterator`. You should not attempt to modify the contents of a font family iterator.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontIterator

An opaque structure used to enumerate fonts.

```
struct FMFontIterator {
    UInt32 reserved[16];
};
typedef struct FMFontIterator FMFontIterator;
```

Fields
`reserved`
Reserved for Apple's use.

Discussion
You initialize a structure of type `FMFontIterator` by calling the function `FMCreateFontIterator`. You should not attempt to modify the contents of a font iterator.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontSize

Represents a font size.

```
typedef SInt16 FMFontSize;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMFontStyle

Represents a font style.

```
typedef SInt16 FMFontStyle;
```

Discussion
The low 8 bits of a Font Manager font style correspond to a QuickDraw style.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

FMGeneration

Keeps track of any operation that adds, deletes, or modifies one or more fonts or font family objects.

```
typedef UInt32 FMGeneration;
```

Discussion
Any operation that adds, deletes, or modifies one or more fonts or font family objects triggers an update of a global generation seed value. Each font and font family modified during a transaction is tagged with a copy of the generation seed.

You can use the function `FMGetGeneration` to get the current value of the generation seed. Then you can use this information in conjunction with the functions `FMGetFontGeneration` and `FMGetFontFamilyGeneration` to identify any changes in the font database.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSUI Data Types

The data types in this section are used by Apple Type Services for Unicode Imaging (ATSUI).

ATSGlyph

Represents a glyph code.

```
typedef UInt16 ATSGlyph;
```

ATSGlyphIdealMetrics

Contains ideal (resolution-independent) metrics for a glyph.

```
struct ATSGlyphIdealMetrics {
    Float32Point advance;
    Float32Point sideBearing;
    Float32Point otherSideBearing;
};
typedef struct ATSGlyphIdealMetrics ATSGlyphIdealMetrics;
```

Fields

`advance`
The amount by which the pen is advanced after drawing the glyph.

`sideBearing`
The offset from the glyph origin to the beginning of the glyph image.

`otherSideBearing`
The offset from the end of the glyph image to the end of the glyph advance.

Discussion
This data structure is passed as a parameter to the ATSUI function `ATSUGlyphGetIdealMetrics`. For more information, see *Inside Mac OS X: ATSUI Reference*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSGlyphRef

Represents a glyph reference.

```
typedef UInt16 ATSGlyphRef;
```

Discussion
This data type is used in the ATSUI data structure `ATSLayoutRecord`. For information, see *Inside Mac OS X: ATSUI Reference*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

ATSGlyphScreenMetrics

Contains device-adjusted font metric information for glyphs in a font.

```
struct ATSGlyphScreenMetrics {
    Float32Point deviceAdvance;
    Float32Point topLeft;
    UInt32 height;
    UInt32 width;
    Float32Point sideBearing;
    Float32Point otherSideBearing;
};
typedef struct ATSGlyphScreenMetrics ATSGlyphScreenMetrics;
```

Fields

`deviceAdvance`
The number of pixels of the advance for the glyph as actually drawn on the screen.

`topLeft`
The top-left point of the glyph in device coordinates.

`height`
The height of the glyph, in pixels. The glyph specified by this value may overlap with other glyphs when drawn.

`width`
The width of the glyph, in pixels. The glyph specified by this value may overlap with other glyphs when drawn.

`sideBearing`
The origin-side bearing, in pixels.

`otherSideBearinge`
The trailing-side bearing, in pixels.

Discussion

The `ATSGlyphScreenMetrics` data structure contains metrics for where glyphs should be drawn on the screen. The metrics include any adjustments needed to display the glyphs properly on the current screen. The structure is returned by the ATSUI function `ATSUGlyphGetScreenMetrics`. Many of the metrics in this structure are `Float32Point` data types so the metrics can integrate with Quartz functions, which all require `Float32Point` data types.

For information on the ATSUI function `ATSUGlyphGetScreenMetrics`, see *Inside Mac OS X: ATSUI Reference*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSTypes.h`

ATSUCurvePath

Contains curve information for a glyph path.

```
struct ATSUCurvePath {
    UInt32 vectors;
    UInt32 controlBits[1];
    Float32Point vector[1];
};
typedef struct ATSUCurvePath ATSUCurvePath;
```

Fields

`vectors`
: The number of values in each of the `controlBits` and `vector` arrays.

`controlBits`
: An array of control bit values that, together with the values in the `vector` array, define one cubic curve in a glyph.

`vector`
: An array of vector values that, together with the values in the `controlBits` array, define one cubic curve in a glyph.

Discussion

This data structure is used in the `ATSUCurvePaths` (page 701) data structure. The `ATSUCurvePaths` data structure is passed as a parameter to the ATSUI function `ATSUGlyphGetCurvePaths`. For more information, see *Inside Mac OS X: ATSUI Reference*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSTypes.h`

ATSUCurvePaths

Contains curve information for an array of glyph paths.

```
struct ATSUCurvePaths {
    UInt32 contours;
    ATSUCurvePath contour[1];
};
typedef struct ATSUCurvePaths ATSUCurvePaths;
```

Fields

`contours`
: The number of cubic curves contained in the `contour` array.

`contour`
: An array of cubic curves that define the outline of a glyph.

Discussion

The `ATSUCurvePaths` data structure is passed as a parameter to the ATSUI function `ATSUGlyphGetCurvePaths`. For more information, see *Inside Mac OS X: ATSUI Reference*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSTypes.h`

GlyphID

Represents a reference to a glyph.

```
typedef ATSGlyphRef GlyphID;
```

Discussion
The `GlyphID` data type is used by ATSUI. For more information, see *Inside Mac OS X: ATSUI Reference*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSTypes.h`

Constants

ATS Constants

Assorted Options

Specify assorted options.

```
enum {
    kATSOptionFlagsDefault = kNilOptions,
    kATSOptionFlagsComposeFontPostScriptName = 1 << 0,
    kATSOptionFlagsUseDataForkAsResourceFork = 1 << 8,
    kATSOptionFlagsUseResourceFork = 2 << 8,
    kATSOptionFlagsUseDataFork = 3 << 8
};
```

Constants

`kATSOptionFlagsDefault`
Specifies to use the default setting.
Available in Mac OS X v10.0 and later.
Declared in `ATSFont.h`.

`kATSOptionFlagsComposeFontPostScriptName`
Specifies the composed PostScript name of a font.
Available in Mac OS X v10.0 and later.
Declared in `ATSFont.h`.

`kATSOptionFlagsUseDataForkAsResourceFork`
Specifies to use the data fork of a font as a resource fork. You can pass this option in the `iOptions` parameter for the function `ATSFontActivateFromFileSpecification` (page 648).
Available in Mac OS X v10.0 and later.
Declared in `ATSFont.h`.

`kATSOptionFlagsUseResourceFork`
Specifies to use the resource fork of a font.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

`kATSOptionFlagsUseDataFork`
Specifies to use the data fork of a font.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

Automatic Activation Settings

Values for automatic activation settings.

```
enum {
  kATSFontAutoActivationDefault = 0,
  kATSFontAutoActivationDisabled = 1,
  kATSFontAutoActivationEnabled = 2,
  kATSFontAutoActivationAsk     = 4
}
typedef UInt32  ATSFontAutoActivationSetting;
```

Constants

`kATSFontAutoActivationDefault`
Resets the setting the the default state. For application settings this clears the setting. For the global setting, it reverts to the initial system setting, `kATSFontAutoActivationEnabled`.

Available in Mac OS X v10.5 and later.

Declared in `ATSFont.h`.

`kATSFontAutoActivationAsk`
Asks the user before automatically activating fonts requested by the application.

Available in Mac OS X v10.5 and later.

Declared in `ATSFont.h`.

`kATSFontAutoActivationEnabled`
Enables automatic activation of fonts.

Available in Mac OS X v10.5 and later.

Declared in `ATSFont.h`.

`kATSFontAutoActivationDisabled`
Disables automatic activation of fonts.

Available in Mac OS X v10.5 and later.

Declared in `ATSFont.h`.

Declared In

`ATSFont.h`

Context Options

Specify a context to use when enumerating, activating, or deactivating fonts and font families.

```
typedef UInt32 ATSFontContext;
enum {
    kATSFontContextUnspecified = 0,
    kATSFontContextGlobal = 1,
    kATSFontContextLocal = 2
};
```

Constants

`kATSFontContextUnspecified`
Indicates a context is not specified. This option has the same result as providing the option `kATSFontContextLocal`.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

`kATSFontContextGlobal`
Specifies to use a global context. Fonts with a global context are available to all applications on the system.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

`kATSFontContextLocal`
Specifies to use a local context. Fonts with a local context are available to your application.

Available in Mac OS X v10.1 and later.

Declared in `ATSFont.h`.

Discussion

Context refers to the font's availability and can be local or global. You provide a context as an option to such functions as `ATSFontActivateFromFileSpecification` (page 648), `ATSFontActivateFromMemory` (page 649), `ATSFontFamilyIteratorCreate` (page 655), `ATSFontFamilyIteratorReset` (page 658). `ATSFontIteratorCreate` (page 671), and `ATSFontIteratorReset` (page 674).

Data Not Specified Constants

Indicate data that is not specified.

```
enum {
    kATSGenerationUnspecified = 0,
    kATSFontContainerRefUnspecified = 0,
    kATSFontFamilyRefUnspecified = 0,
    kATSFontRefUnspecified = 0
};
```

Constants

`kATSGenerationUnspecified`
Indicates the generation is not specified.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

`kATSFontContainerRefUnspecified`
Indicates the font container reference is not specified.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

`kATSFontFamilyRefUnspecified`
Indicates the font family reference is not specified.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

`kATSFontRefUnspecified`
Indicates the font reference is not specified.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

Discussion
You can pass these constants to functions when you either don't know the appropriate value or do not care to obtain the associated information. These constants can also be returned to you to indicate an error.

Font Filter Selectors

Specify the type of criteria to use when limiting an iteration.

```
enum ATSFontFilterSelector {
    kATSFontFilterSelectorUnspecified = 0,
    kATSFontFilterSelectorGeneration = 3,
    kATSFontFilterSelectorFontFamily = 7,
    kATSFontFilterSelectorFontFamilyApplierFunction = 8,
    kATSFontFilterSelectorFontApplierFunction = 9
};
typedef enum ATSFontFilterSelector ATSFontFilterSelector;
```

Constants
`kATSFontFilterSelectorUnspecified`
Specifies to limit an iteration based on unspecified criteria. In this case, the default is used, which is to iterate using a local context with an unrestricted scope.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

`kATSFontFilterSelectorGeneration`
Specifies to limit an iteration based on generation criteria.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

`kATSFontFilterSelectorFontFamily`
Specifies to limit an iteration based on font family criteria.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

`kATSFontFilterSelectorFontFamilyApplierFunction`
Specifies to limit an iteration based on criteria defined by a font family applier function.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

`kATSFontFilterSelectorFontApplierFunction`
Specifies to limit an iteration based on criteria defined by a font applier function.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

Discussion
You use these constants in the data structure `ATSFontFilter` (page 688) to specify the type of data in the `filter` union.

Font Filter Versions

Specify the version of a font filter.

```
typedef UInt32 ATSFontFormat;
enum {
    kATSFontFilterCurrentVersion = 0
};
```

Constants
`kATSFontFilterCurrentVersion`
Specifies to use the current version of a font filter.

Available in Mac OS X v10.0 and later.

Declared in `ATSFont.h`.

Discussion
There is currently only one constant in this enumeration. You can assign this constant to the `version` field in the `ATSFontFilter` (page 688) data structure.

Font Formats

Specify a font format.

```
enum {
    kATSFontFormatUnspecified = 0
};
```

Constants
`kATSFontFormatUnspecified`
Indicates the font format is not specified. You can pass this in the `iFormat` parameter of the function `ATSFontActivateFromFileSpecification` (page 648).

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

Discussion
There are no other font formats currently defined for this enumeration.

Font Request Query Keys

Represent keys in a font request query dictionary.

```
#define kATSQueryClientPID                                              CFSTR("ATS
 client
pid")
#define kATSQueryQDFamilyName               CFSTR("font family
name")
#define kATSQueryFontName                CFSTR("font name")
#define kATSQueryFontPostScriptName      CFSTR("font PS name")
#define kATSQueryFontNameTableEntries    CFSTR("font name table
entries")
#define kATSFontNameTableCode            CFSTR("font name code")
#define kATSFontNameTablePlatform        CFSTR("font platform
code")
#define kATSFontNameTableScript          CFSTR("font script
code")
#define kATSFontNameTableLanguage        CFSTR("font language
code")
#define kATSFontNameTableBytes
CFSTR("font
name table bytes")
```

Constants

`kATSQueryClientPID`
Specifies a process ID. The value associated with this key is a `CFNumberRef` value that refers to a the process ID (`pid_t`) of the application making the query.

`kATSQueryQDFamilyName`
Specifies a QuickDraw family name. The value associated with this key is a `CFStringRef` value that refers to the QuickDraw family name of the requested font. For example, the name passed to the function `GetFNum`.

`kATSQueryFontName`
Specifies a font name. The value associated with this key is a `CFStringRef` value that refers to the full name of the requested font. You can use this font name as an argument to the function `ATSFontFindFromName` (page 660).

`kATSQueryFontPostScriptName`
Specifies the PostScript name of a font. The value associated with this key is a `CFStringRef` value that refers to either the PostScript name derived from the font's `FOND` resource or from the font's `'sfnt'` name table, with preference given to the `FOND` PostScript name. You can use this font name as an argument to the function `ATSFontFindFromPostScriptName` (page 661).

`kATSQueryFontNameTableEntries`
Specifies the descriptor for `'sfnt'` name table entries. The value associated with this key an array (`CFArrayRef`) of `CFDictionaryRef` values that describe entries in a name table. A font must have all of the specified entries to be considered a match.

`kATSFontNameTableCode`
Specifies the font name's name code. The value associated with this key is a `CFNumberRef`. If no value is specified, the value `kFontNoNameCode` is used.

`kATSFontNameTablePlatform`
Specifies the font name's platform code. The value associated with this key is a `CFNumberRef`. If no value is specified, the value `kFontNoPlatformCode` is used.

`kATSFontNameTableScript`
Specifies the font name's script code. The value associated with this key is a `CFNumberRef`. If no value is specified, the value `kFontNoScriptCode` is used.

`kATSFontNameTableLanguage`
: Specifies the font name's language code. The value associated with this key is a `CFNumberRef`. If no value is specified, the value `kFontNoLanguageCode` is used.

`kATSFontNameTableBytes`
: Specifies the raw bytes of the font name. The value associated with this key is a `CFDataRef` value that refers to the raw name bytes for the font.

Discussion
Font request query keys appear in the dictionary passed to, and returned by, your `ATSFontQueryCallback` (page 683) callback function. The keys comprise a property list (`CFPropertyList`) that defines the query sent to your callback. On return, you supply a property list that specifies your response to the query.

Font Query Message ID

Specifies a message ID for a font request query.

```
enum ATSFontQueryMessageID {
    kATSQueryActivateFontMessage = 'atsa'
};
typedef enum ATSFontQueryMessageID ATSFontQueryMessageID;
```

Constants
`kATSQueryActivateFontMessage`
: Specifies to activate a font message. The data associated with this message ID is a flattened `CFDictionaryRef`. The `CFDictionary` contains on or more of the keys described in "Font Request Query Keys" (page 706).

Available in Mac OS X v10.2 and later.

Declared in `ATSFont.h`.

Discussion
There is currently only one constant in this enumeration. You use a constant of this type when you create an `ATSFontQueryCallback` (page 683) callback function.

Iteration Precedence Options

Specify the order of an iteration.

```
enum {
    kATSOptionFlagsIterateByPrecedenceMask = 0x00000001 <<  5
};
```

Constants
`kATSOptionFlagsIterateByPrecedenceMask`
: Specifies to iterate fonts in the order dictated by a precedence mask.

Available in Mac OS X v10.1 and later.

Declared in `ATSFont.h`.

Notification Actions

Specify a notification action.

```
enum ATSFontNotifyAction {
    kATSFontNotifyActionFontsChanged = 1,
    kATSFontNotifyActionDirectoriesChanged = 2
};
typedef enum ATSFontNotifyAction ATSFontNotifyAction;
```

Constants

`kATSFontNotifyActionFontsChanged`

Specifies that your application has activated or deactivated fonts. Typically you call the functions `ATSFontActivateFromFileSpecification` (page 648) or `ATSFontDeactivate` (page 651) multiple times to activate and deactivate fonts. In each call, you set the `iOptions` parameter to `kATSOptionFlagsDoNotNotify` set. When you are done activating and deactivating fonts you can call the function `ATSFontNotify` (page 676) with the `action` parameter set to `kATSFontNotifyActionFontsChanged`. Then ATS notifies all applications who subscribe to notifications of the changes you made.

Available in Mac OS X v10.2 and later.

Declared in `ATSFont.h`.

`kATSFontNotifyActionDirectoriesChanged`

Specifies that your application has made changes to one or more of the font directories. When you are making changes to font directories, you can call the function `ATSFontNotify` (page 676) with the `action` parameter set to `kATSFontNotifyActionDirectoriesChanged`. Then ATS scans these directories and notifies all applications who subscribe to notifications of the changes you made.

Available in Mac OS X v10.2 and later.

Declared in `ATSFont.h`.

Discussion

You can use these options with the function `ATSFontNotify` (page 676).

Notification Options

Specify when ATS should notify your application of changes in the font database.

```
enum ATSFontNotifyOption {
    kATSFontNotifyOptionDefault = 0,
    kATSFontNotifyOptionReceiveWhileSuspended = 1L << 0
};
typedef enum ATSFontNotifyOption ATSFontNotifyOption;
```

Constants

`kATSFontNotifyOptionDefault`

Specifies to use the default behavior of the function `ATSFontNotificationSubscribe` (page 675).

Available in Mac OS X v10.2 and later.

Declared in `ATSFont.h`.

`kATSFontNotifyOptionReceiveWhileSuspended`

Specifies to receive notifications even if the application is in the background. Setting this option can degrade performance; you should set this option if your application is a faceless process or a tool that performs font management functions and requires immediate notification when fonts change.

Available in Mac OS X v10.2 and later.

Declared in `ATSFont.h`.

Discussion
You use notification options when you call the function `ATSFontNotificationSubscribe` (page 675). The default behavior is for applications to receive ATS notifications only when the application runs in the foreground. By default, if the application is suspended, the notification is delivered when the application comes to the foreground.

Scoping Options

Specify the scope to which an operation should apply or a notification schedule.

```
enum {
    kATSOptionFlagsDoNotNotify = 0x00000001 << 8,
    kATSOptionFlagsIterationScopeMask = 0x00000007 << 12,
    kATSOptionFlagsDefaultScope = 0x00000000 << 12,
    kATSOptionFlagsUnRestrictedScope = 0x00000001 << 12,
    kATSOptionFlagsRestrictedScope = 0x00000002 << 12,
    kATSOptionFlagsProcessSubdirectories = 0x00000001 << 6
};
```

Constants

`kATSOptionFlagsDoNotNotify`
Specifies not to send a notification after a font is activated or deactivated globally. You can set the `iOptions` parameter of the functions `ATSFontActivateFromFileSpecification` (page 648) or `ATSFontDeactivate` (page 651) to this constant. When you are done activating and deactivating fonts you can call the function `ATSFontNotify` (page 676) with the `action` parameter set to `kATSFontNotifyActionFontsChanged`. Then ATS notifies all applications who subscribe to notifications of the changes you made.

Available in Mac OS X v10.2 and later.

Declared in `ATSFont.h`.

`kATSOptionFlagsIterationScopeMask`
Specifies mask option bits 12-14 for iteration scopes.

Available in Mac OS X v10.1 and later.

Declared in `ATSFont.h`.

`kATSOptionFlagsDefaultScope`
Specifies to use the default scope, which is equivalent to `kATSOptionFlagsUnRestrictedScope`. You can pass this as a parameter to the functions `ATSFontFamilyIteratorCreate` (page 655), `ATSFontFamilyIteratorReset` (page 658), `ATSFontIteratorCreate` (page 671) and `ATSFontIteratorReset` (page 674).

Available in Mac OS X v10.1 and later.

Declared in `ATSFont.h`.

`kATSOptionFlagsUnRestrictedScope`
Specifies to use an unrestricted scope. You can pass this as a parameter to the functions `ATSFontFamilyIteratorCreate` (page 655), `ATSFontFamilyIteratorReset` (page 658), `ATSFontIteratorCreate` (page 671) and `ATSFontIteratorReset` (page 674).

Available in Mac OS X v10.1 and later.

Declared in `ATSFont.h`.

`kATSOptionFlagsRestrictedScope`
: Specifies to use a restricted scope. You can pass this as a parameter to the functions `ATSFontFamilyIteratorCreate` (page 655), `ATSFontFamilyIteratorReset` (page 658), `ATSFontIteratorCreate` (page 671) and `ATSFontIteratorReset` (page 674).

Available in Mac OS X v10.1 and later.

Declared in `ATSFont.h`.

`kATSOptionFlagsProcessSubdirectories`
: Specifies to process the font directories within a font directory. You can pass this as a parameter to the function `ATSFontActivateFromFileSpecification` (page 648).

Available in Mac OS X v10.2 and later.

Declared in `ATSFont.h`.

Discussion
Scope refers to whether a font's use is restricted or unrestricted. Fonts with a restricted scope can be used only by your application whereas fonts with an unrestricted scope cay be used by all applications.

Font Manager Constants

FM Filter Format

Specifies a filter format.

```
enum {
    kFMCurrentFilterFormat = 0
};
```

Constants
`kFMCurrentFilterFormat`
: Specifies the current filter format. You can use this to set the format field when you initialize the `FMFilter` data type for use in creating an iterator object with the functions `FMCreateFontFamilyIterator` or `FMCreateFontIterator`. Currently, this is the only format you can specify.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

FM Filter Selectors

Specifies a filter type.

```
typedef UInt32 FMFilterSelector;
enum {
    kFMFontTechnologyFilterSelector = 1,
    kFMFontContainerFilterSelector = 2,
    kFMGenerationFilterSelector = 3,
    kFMFontFamilyCallbackFilterSelector = 4,
    kFMFontCallbackFilterSelector = 5,
    kFMFontDirectoryFilterSelector = 6
};
```

Constants

`kFMFontTechnologyFilterSelector`
Selects font technology filter. You can use this filter only with a font iterator.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kFMFontContainerFilterSelector`
Selects font container filter. You can use this filter only with a font iterator.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kFMGenerationFilterSelector`
Selects generation filter. You can use this filter only with a font family iterator.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kFMFontFamilyCallbackFilterSelector`
Indicates a custom filter to be used only with a font family iterator.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kFMFontCallbackFilterSelector`
Indicates a custom filter to be used only with a font iterator.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

Discussion

You use these constants to specify a filter type in the `FMFilter` (page 693) data structure used by many Font Manager functions.

FM Font Technologies

Specify a font technology.

```
enum {
    kFMTrueTypeFontTechnology = 'true',
    kFMPostScriptFontTechnology = 'typ1'
};
```

Constants

`kFMTrueTypeFontTechnology`
Indicates True Type font technology.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kFMPostScriptFontTechnology`
Indicates Post Script font technology.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

Invalid Values

Specify an invalid value.

```
enum {
    kInvalidGeneration = 0,
    kInvalidFontFamily = -1,
    kInvalidFont = 0
};
```

Constants

`kInvalidGeneration`
Indicates an invalid generation value.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kInvalidFontFamily`
Indicates the font family reference is invalid.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kInvalidFont`
Indicates the font reference is invalid.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

Discussion

The `kInvalidGeneration`, `kInvalidFontFamily`, and `kInvalidFont` constants may be used to indicate invalid values for generation count, font family, and font data types.

ATSUI Constants

Convenience Constants

Represent numerical values that are commonly used in font calculations.

```
enum {
    kATSItalicQDSkew = (1 << 16) / 4,
    kATSBoldQDStretch = (1 << 16) * 3 / 2,
    kATSRadiansFactor = 1144
};
```

Constants

`kATSItalicQDSkew`
A `Fixed` value of `0.25` that represents the skew used by QuickDraw to draw italicized glyphs.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

`kATSBoldQDStretch`
A `Fixed` value that represents the stretch-factor used by QuickDraw to draw bold-faced glyphs.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

`kATSRadiansFactor`
A `Fixed` value of approximately `pi/180(0.0174560546875)` that represents an angle of 1 radian. This is a convenience constant you can use when you draw rotated text.

Available in Mac OS X v10.0 and later.

Declared in `ATSTypes.h`.

Discussion

These constants are provided for convenience. Your application can use them when it needs to perform font calculations.

Version Notes

Available beginning with ATSUI 1.0.

Curve Types

Specify a curve type used to draw a font.

```
typedef UInt16 ATSCurveType;
enum {
    kATSCubicCurveType = 0x0001,
    kATSQuadCurveType = 0x0002,
    kATSOtherCurveType = 0x0003
};
```

Constants

`kATSCubicCurveType`
Specifies a cubic curve.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kATSQuadCurveType`
Specifies a quadratic curve.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

`kATSOtherCurveType`
Specifies a curve other than cubic or quadratic.
Available in Mac OS X v10.0 and later.
Declared in `ATSTypes.h`.

Discussion

These are used in the ATSUI function `ATSUGetNativeCurveType`. See *Inside Mac OS X: ATSUI Reference* for more information.

Deleted Glyph Code

Specifies that a glyph is deleted.

```
enum {
    kATSDeletedGlyphcode = 0xFFFF
};
```

Constants

`kATSDeletedGlyphcode`
Indicates that a glyph is deleted. That is, the glyph is set to no longer appear in a text layout.
Available in Mac OS X v10.2 and later.
Declared in `ATSTypes.h`.

Discussion

This constant is used by ATSUI. When a glyph is deleted, ATSUI sets the corresponding `ATSGlyphRef` (page 699) to `kATSDeletedGlyphcode`. For more information, see *Inside Mac OS X: ATSUI Reference*.

Result Codes

The most common result codes returned by Apple Type Services for Fonts are listed below.

Result Code	Value	Description
kATSIterationCompleted	-980L	The iteration is complete. Available in Mac OS X v10.0 and later.
kATSInvalidFontFamilyAccess	-981L	Your application tried to access an invalid font family. Available in Mac OS X v10.0 and later.
kATSInvalidFontAccess	-982L	Your application tried to access an invalid font. Available in Mac OS X v10.0 and later.
kATSIterationScopeModified	-983L	The font database changed during an iteration. Available in Mac OS X v10.0 and later.
kATSInvalidFontTableAccess	-984L	Your application tried to access an invalid font table. Available in Mac OS X v10.0 and later.
kATSInvalidFontContainerAccess	-985L	Your application tried to access an invalid font container. Available in Mac OS X v10.0 and later.

ColorSync Manager Reference

Framework: ApplicationServices/ApplicationServices.h, Carbon/Carbon.h

Declared in CMApplication.h
CMMComponent.h
CMTypes.h
CMCalibrator.h
CMScriptingPlugin.h
CMDeviceIntegration.h

Overview

The ColorSync Manager is the API for ColorSync, a platform-independent color management system from Apple. ColorSync provides essential services for fast, consistent, and accurate color calibration, proofing, and reproduction using input, output, and display devices. ColorSync also provides an interface to system-wide color management settings that allows users to save color settings for specific jobs and switch between settings.

You need this reference if your software product performs color drawing, printing, or calculation, or if your peripheral device supports color. You also need this reference if you are creating a color management module (CMM)—a component that implements color-matching, color-conversion, and gamut-checking services.

The Color Picker Manager, documented separately, provides a standard user interface for soliciting color choices.

Carbon supports the majority of the ColorSync Manager programming interface. However, ColorSync 1.0 compatibility calls such as `CWNewColorWorld`, `GetProfile`, and `SetProfile` are not supported.

Nor does Carbon support ColorSync functions used for color management modules (CMMs). These functions aren't supported because Mac OS X uses Bundle Services to implement CMMs.

Some applications use the Component Manager to determine what CMMs are available. You cannot use the Component Manager for this purpose in Mac OS X. Apple has, however, provided a the function `CMIterateCMMInfo` to query for available CMMs.

Functions by Task

Accessing Profiles

`CMOpenProfile` (page 790)
Opens the specified profile and returns a reference to the profile.

`CMValidateProfile` (page 818)
Indicates whether the specified profile contains the minimum set of elements required by the current color management module (CMM) for color matching or color checking.

`CMCloseProfile` (page 728)
Decrements the reference count for the specified profile reference and, if the reference count reaches 0, frees all private memory and other resources associated with the profile.

`CMUpdateProfile` (page 816)
Saves modifications to the specified profile.

`CMCopyProfile` (page 740)
Duplicates the specified existing profile.

`CMProfileModified` (page 795)
Indicates whether the specified profile has been modified since it was created or last updated.

`CMGetProfileMD5` (page 771)
Gets the MD5 checksum from the profile header (message digest)

`CMGetProfileHeader` (page 769)
Obtains the profile header for the specified profile.

`CMSetProfileHeader` (page 813)
Sets the header for the specified profile.

`NCMGetProfileLocation` (page 841)
Obtains either a profile location structure for a specified profile or the size of the location structure for the profile.

`CMCloneProfileRef` (page 727)
Increments the reference count for the specified profile reference.

`CMGetProfileRefCount` (page 772)
Obtains the current reference count for the specified profile.

`CMFlattenProfile` (page 748) Deprecated in Mac OS X v10.5
Transfers a profile stored in an independent disk file to an external profile format that can be embedded in a graphics document.

`CMGetProfileLocation` (page 770) Deprecated in Mac OS X v10.5
Obtains the location of a profile based on the specified profile reference.

`NCMUnflattenProfile` (page 843) Deprecated in Mac OS X v10.5
Unflattens a previouslyflattened profile.

Iterating Installed Profiles

`CMIterateColorSyncFolder` (page 780)
Iterates over the available profiles.

`CMGetColorSyncFolderSpec` (page 749) Deprecated in Mac OS X v10.5
Obtains the volume reference number and the directory ID for a ColorSync Profiles folder.

Creating Profiles

`CMNewProfile` (page 788)
Creates a new profile and associated backing copy.

`NCWNewLinkProfile` (page 848)
Obtains a profile reference for the specified by the profile location.

`CMMakeProfile` (page 784)
Makes a display or abstract profile by modifying an existing one.

`CWNewLinkProfile` (page 830) Deprecated in Mac OS X v10.5
Creates a device link profile based on the specified set of profiles.

Accessing Special Profiles

`CMGetSystemProfile` (page 778)
Obtains a reference to the current system profile.

`CMGetDefaultProfileBySpace` (page 752)
Gets the default profile for the specified color space.

`CMGetDefaultProfileByUse` (page 753)
Obtains the users' preferred device profile setting.

`CMGetProfileByAVID` (page 767)
Gets the current profile for a monitor.

`CMSetProfileByAVID` (page 808)
Sets the profile for the specified monitor, optionally setting video card gamma.

`CMSetDefaultProfileBySpace` (page 801) Deprecated in Mac OS X v10.5
Sets the default profile for the specified color space.

`CMSetDefaultProfileByUse` (page 801) Deprecated in Mac OS X v10.5
Sets values for device profile settings.

`CMSetSystemProfile` (page 814) Deprecated in Mac OS X v10.5
Sets the current system profile.

`NCMSetSystemProfile` (page 842) Deprecated in Mac OS X v10.5
Sets the location of a color profile.

Accessing Profile Elements

`CMCountProfileElements` (page 744)
Counts the number of elements in the specified profile.

`CMProfileElementExists` (page 792)
Tests whether the specified profile contains a specific element based on the element's tag signature.

`CMGetProfileElement` (page 768)
Obtains element data from the specified profile based on the specified element tag signature.

`CMSetProfileElement` (page 810)
Sets or replaces the element data for a specific tag in the specified profile.

`CMSetProfileElementSize` (page 812)
Reserves the element data size for a specific tag in the specified profile before setting the element data.

`CMGetPartialProfileElement` (page 765)
Obtains a portion of the element data from the specified profile based on the specified element tag signature.

`CMSetPartialProfileElement` (page 807)
Sets part of the element data for a specific tag in the specified profile.

`CMGetIndProfileElementInfo` (page 761)
Obtains the element tag and data size of an element by index from the specified profile.

`CMGetIndProfileElement` (page 760)
Obtains the element data corresponding to a particular index from the specified profile.

`CMSetProfileElementReference` (page 811)
Adds a tag to the specified profile to refer to data corresponding to a previously set element.

`CMRemoveProfileElement` (page 797)
Removes an element corresponding to a specific tag from the specified profile.

Accessing Profile Descriptions

`CMCopyProfileDescriptionString` (page 742)
Returns the name of a profile as a CFString.

`CMCopyProfileLocalizedString` (page 742)
Gets one specific string out of a profile

`CMCopyProfileLocalizedStringDictionary` (page 743)
Obtains a CFDictionary which contains the language locale and string for multiple localizations from a given tag.

`CMSetProfileLocalizedStringDictionary` (page 813)
Writes a dictionary of localized strings to a given tag in a profile.

`CMGetProfileDescriptions` (page 767)
Obtains the description tag data for a specified profile.

`CMSetProfileDescriptions` (page 809)
Sets the description tag data for a specified profile.

`CMGetScriptProfileDescription` (page 777) Deprecated in Mac OS X v10.5
Obtains the internal name (or description) of a profile and the script code identifying the language in which the profile name is specified from the specified profile.

Accessing Name-Class Profiles

`CMGetNamedColorInfo` (page 763)
Obtains information about a named color space from its profile reference.

`CMGetNamedColorValue` (page 764)
Obtains device and PCS color values for a specific color name from a named color space profile.

`CMGetIndNamedColorValue` (page 759)
Obtains device and PCS color values for a specific named color index from a named color space profile.

`CMGetNamedColorIndex` (page 762)
Obtains a named color index for a specific color name from a named color space profile.

`CMGetNamedColorName` (page 763)
Obtains a named color name for a specific named color index from a named color space profile.

Working With ColorWorlds

`NCWNewColorWorld` (page 846)
Creates a color world for color matching based on the specified source and destination profiles.

`CWConcatColorWorld` (page 823)
Sets up a color world that includes a set of profiles for various color transformations among devices in a sequence.

`NCWConcatColorWorld` (page 845)
Defines a color world for color transformations among a series of concatenated profiles.

`CWDisposeColorWorld` (page 825)
Releases the private storage associated with a color world when your application has finished using the color world.

`CWMatchColors` (page 828)
Matches colors in a color list, using the specified color world.

`CWCheckColors` (page 821)
Tests a list of colors using a specified color world to see if they fall within the gamut of a destination device.

`CWMatchBitmap` (page 826)
Matches the colors of a bitmap to the gamut of a destination device using the profiles specified by a color world.

`CWCheckBitmap` (page 819)
Tests the colors of the pixel data of a bitmap to determine whether the colors map to the gamut of the destination device.

`CWFillLookupTexture` (page 826)
Fills a 3-D lookup texture from a color world.

`CMGetCWInfo` (page 751) Deprecated in Mac OS X v10.5
Obtains information about the color management modules (CMMs) used for a specific color world.

Converting Colors

`CMConvertFixedXYZToXYZ` (page 730) Deprecated in Mac OS X v10.5
Converts colors specified in XYZ color space whose components are expressed as Fixed XYZ 32-bit signed values of type `CMFixedXYZColor` to equivalent colors expressed as XYZ 16-bit unsigned values of type `CMXYZColor`.

`CMConvertHLSToRGB` (page 730) Deprecated in Mac OS X v10.5
Converts colors specified in the HLS color space to equivalent colors defined in the RGB color space.

`CMConvertHSVToRGB` (page 731) Deprecated in Mac OS X v10.5
Converts colors specified in the HSV color space to equivalent colors defined in the RGB color space.

`CMConvertLabToXYZ` (page 732) Deprecated in Mac OS X v10.5
Converts colors specified in the L*a*b* color space to the XYZ color space.

`CMConvertLuvToXYZ` (page 733) Deprecated in Mac OS X v10.5
Converts colors specified in the L*u*v* color space to the XYZ color space.

`CMConvertRGBToGray` (page 733) Deprecated in Mac OS X v10.5
Converts colors specified in the RGB color space to equivalent colors defined in the Gray color space.

CMConvertRGBToHLS (page 734) Deprecated in Mac OS X v10.5
Converts colors specified in the RGB color space to equivalent colors defined in the HLS color space.

CMConvertRGBToHSV (page 735) Deprecated in Mac OS X v10.5
Converts colors specified in the RGB color space to equivalent colors defined in the HSV color space when the device types are the same.

CMConvertXYZToFixedXYZ (page 736) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space whose components are expressed as XYZ 16-bit unsigned values of type CMXYZColor to equivalent colors expressed as 32-bit signed values of type CMFixedXYZColor.

CMConvertXYZToLab (page 736) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space to the L*a*b* color space.

CMConvertXYZToLuv (page 737) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space to the L*u*v* color space.

CMConvertXYZToXYZ (page 738) Deprecated in Mac OS X v10.5
Converts a source color to a destination color using the specified chromatic adaptation method.

CMConvertXYZToYxy (page 739) Deprecated in Mac OS X v10.5
Converts colors specified in the XYZ color space to the Yxy color space.

CMConvertYxyToXYZ (page 739) Deprecated in Mac OS X v10.5
Converts colors specified in the Yxy color space to the XYZ color space.

Working With CMMs

CMIterateCMMInfo (page 779)
Iterates through the color management modules installed on the system.

CMGetPreferredCMM (page 766) Deprecated in Mac OS X v10.5
Identifies the preferred CMM specified by the ColorSync control panel.

Working With PostScript

CMGetPS2ColorSpace (page 776)
Obtains color space element data in text format usable as the parameter to the PostScript setColorSpace operator, which characterizes the color space of subsequent graphics data.

CMGetPS2ColorRenderingIntent (page 774)
Obtains the rendering intent element data in text format usable as the parameter to the PostScript findRenderingIntent operator, which specifies the color-matching option for subsequent graphics data.

CMGetPS2ColorRendering (page 773)
Obtains the color rendering dictionary (CRD) element data usable as the parameter to the PostScript setColorRendering operator, which specifies the PostScript color rendering dictionary to use for the following graphics data.

CMGetPS2ColorRenderingVMSize (page 775)
Determines the virtual memory size of the color rendering dictionary (CRD) for a printer profile before your application or driver obtains the CRD and sends it to the printer.

Working With QuickDraw

`CMEnableMatchingComment` (page 746) Deprecated in Mac OS X v10.4
Inserts a comment into the currently open picture to turn matching on or off.

`CMEndMatching` (page 747) Deprecated in Mac OS X v10.4
Concludes a QuickDraw-specific ColorSync matching session initiated by a previous call to the `NCMBeginMatching` function.

`CWCheckPixMap` (page 822) Deprecated in Mac OS X v10.4
Checks the colors of a pixel map using the profiles of a specified color world to determine whether the colors are in the gamut of the destination device.

`CWMatchPixMap` (page 829) Deprecated in Mac OS X v10.4
Matches a pixel map in place based on a specified color world.

`NCMBeginMatching` (page 838) Deprecated in Mac OS X v10.4
Sets up a QuickDraw-specific ColorSync matching session, using the specified source and destination profiles.

`NCMDrawMatchedPicture` (page 840) Deprecated in Mac OS X v10.4
Matches a picture's colors to a destination device's color gamut, as the picture is drawn, using the specified destination profile.

`NCMUseProfileComment` (page 843) Deprecated in Mac OS X v10.4
Automatically embeds a profile or a profile identifier into an open picture.

Registering Devices

`CMRegisterColorDevice` (page 797)
Registers a device with ColorSync.

`CMUnregisterColorDevice` (page 815)
Unregisters a device.

Accessing Default Devices

`CMGetDefaultDevice` (page 752)
Gets the default device.

`CMSetDefaultDevice` (page 800)
Sets the default device.

Accessing Devices Profiles

`CMGetDeviceFactoryProfiles` (page 754)
Retrieves the original profiles for a given device.

`CMSetDeviceFactoryProfiles` (page 803)
Establishes the profiles used by a given device.

`CMGetDeviceDefaultProfileID` (page 754)
Gets the default profile ID for a given device.

`CMSetDeviceDefaultProfileID` (page 802)
Sets the default profile ID for a given device.

`CMSetDeviceProfile` (page 803)
Change the profile used by a given device.

`CMGetDeviceProfile` (page 756)
Gets a profile used by a given device.

`CMGetDeviceProfiles` (page 756) Deprecated in Mac OS X v10.5
Gets the profiles used by a given device.

`CMSetDeviceProfiles` (page 804) Deprecated in Mac OS X v10.5
Changes the profiles used by a given device.

Accessing Device State and Information

`CMGetDeviceState` (page 757)
Gets the state of a device.

`CMSetDeviceState` (page 805)
Sets the state of a device.

`CMGetDeviceInfo` (page 755)
Gets information about a specified device.

Iterating Over Devices and Device Profiles

`CMIterateColorDevices` (page 780)
Iterates through the color devices available on the system, returning device information to a callback you supply.

`CMIterateDeviceProfiles` (page 782)
Iterates through the device profiles available on the system and returns information about profiles of the devices to a callback you supply.

Working With Image Files

`CMCountImageProfiles` (page 743) Deprecated in Mac OS X v10.5
Obtains a count of the number of embeded profiles for a given image.

`CMEmbedImage` (page 746) Deprecated in Mac OS X v10.5
Embeds an image with an ICC profile.

`CMGetImageSpace` (page 758) Deprecated in Mac OS X v10.5
Returns the signature of the data color space in which the color values of colors in an image are expressed.

`CMGetIndImageProfile` (page 758) Deprecated in Mac OS X v10.5
Obtains a specific embeded profile for a given image.

`CMLinkImage` (page 783) Deprecated in Mac OS X v10.5
Matches an image file with a device link profile.

`CMMatchImage` (page 787) Deprecated in Mac OS X v10.5
Color matches an image file.

`CMProofImage` (page 796) Deprecated in Mac OS X v10.5
Proofs an image.

`CMSetIndImageProfile` (page 807) Deprecated in Mac OS X v10.5
Sets a specific embeded profile for a given image.

`CMUnembedImage` (page 814) Deprecated in Mac OS X v10.5
Removes any ICC profiles embeded in an image.

`CMValidImage` (page 819) Deprecated in Mac OS X v10.5
Validates the specified image file.

Working With Video Card Lookup Tables

`CMGetGammaByAVID` (page 757)
Obtains the gamma value for the specified display device.

`CMSetGammaByAVID` (page 806)
Sets the gamma for the specified display device.

Miscellaneous

`CMGetColorSyncVersion` (page 750)
Gets ColorSync version information.

`CMLaunchControlPanel` (page 783)
Launches the ColorSync preferences pane.

`CMCalibrateDisplay` (page 727)
Calibrates a display.

Working With Universal Procedure Pointers

`DisposeCMBitmapCallBackUPP` (page 832) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a bitmap callback.

`DisposeCMConcatCallBackUPP` (page 832) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback.

`DisposeCMFlattenUPP` (page 833) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a data-flattening callback.

`DisposeCMMIterateUPP` (page 833) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function.

`DisposeCMProfileAccessUPP` (page 834) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a profile-access callback.

`DisposeCMProfileFilterUPP` (page 834) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a profile-filter callback.

`DisposeCMProfileIterateUPP` (page 834) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer (UPP) to a profile-iteration callback.

`InvokeCMBitmapCallBackUPP` (page 835) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a bitmap callback.

`InvokeCMConcatCallBackUPP` (page 835) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a progress-monitoring callback.

`InvokeCMFlattenUPP` (page 836) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a data-flattening callback.

`InvokeCMMIterateUPP` (page 836) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a a progress-monitoring callback for the `CMIterateCMMInfo` function.

`InvokeCMProfileAccessUPP` (page 837) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a profile-access callback.

`InvokeCMProfileFilterUPP` (page 837) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a profile-filter callback.

`InvokeCMProfileIterateUPP` (page 837) Deprecated in Mac OS X v10.5
Invokes a universal procedure pointer (UPP) to a profile-iteration callback.

`NewCMBitmapCallBackUPP` (page 848) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a bitmap callback.

`NewCMConcatCallBackUPP` (page 849) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a progress-monitoring callback.

`NewCMFlattenUPP` (page 849) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a data-flattening callback.

`NewCMMIterateUPP` (page 850) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function.

`NewCMProfileAccessUPP` (page 850) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a profile-access callback.

`NewCMProfileFilterUPP` (page 851) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a profile-filter callback.

`NewCMProfileIterateUPP` (page 851) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a profile-iteration callback.

Not Recommended

`CMCreateProfileIdentifier` (page 745) Deprecated in Mac OS X v10.5
Creates a profile identifier for a specified profile.

`CMDisposeProfileSearch` (page 745) Deprecated in Mac OS X v10.5
Frees the private memory allocated for a profile search after your application has completed the search.

`CMNewProfileSearch` (page 789) Deprecated in Mac OS X v10.5
Searches the ColorSync Profiles folder and returns a list of 2.x profiles that match the search specification.

`CMProfileIdentifierFolderSearch` (page 792) Deprecated in Mac OS X v10.5
: Searches the ColorSync Profiles folder and returns a list of profile references, one for each profile that matches the specified profile identifier.

`CMProfileIdentifierListSearch` (page 793) Deprecated in Mac OS X v10.5
: Searches a list of profile references and returns a list of all references that match a specified profile identifier.

`CMSearchGetIndProfile` (page 798) Deprecated in Mac OS X v10.5
: Opens the profile corresponding to a specific index into a specific search result list and obtains a reference to that profile.

`CMSearchGetIndProfileFileSpec` (page 799) Deprecated in Mac OS X v10.5
: Obtains the file specification for the profile at a specific index into a search result.

`CMUpdateProfileSearch` (page 817) Deprecated in Mac OS X v10.5
: Searches the ColorSync Profiles folder and updates an existing search result obtained originally from the `CMNewProfileSearch` function.

Functions

CMCalibrateDisplay

Calibrates a display.

```
OSErr CMCalibrateDisplay (
   CalibratorInfo *theInfo
);
```

Parameters

theInfo
: A pointer to a calibrator info data structure that contains the necessary data for calibrating a display.

Return Value

An `OSErr` value.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`CMCalibrator.h`

CMCloneProfileRef

Increments the reference count for the specified profile reference.

```
CMError CMCloneProfileRef (
   CMProfileRef prof
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile whose reference count is incremented.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The ColorSync Manager keeps an internal reference count for each profile reference returned from a call to the `CMOpenProfile`, `CMNewProfile`, or `CMCopyProfile` functions. Calling the `CMCloneProfileRef` function increments the count; calling the function `CMCloseProfile` (page 728) decrements it. The profile remains open as long as the reference count is greater than 0, indicating that at least one routine retains a reference to the profile. When the count reaches 0, the ColorSync Manager releases all private memory, files, or resources allocated in association with that profile.

When your application creates a copy of an entire profile with `CMCopyProfile`, the copy has its own reference count. The `CMCloseProfile` routine should be called for the copied profile, just as for the original. When the reference count reaches 0, private resources associated with the copied profile are freed.

When your application merely duplicates a profile reference, as it may do to pass a profile reference to a synchronous or an asynchronous task, it should call `CMCloneProfileRef` to increment the reference count. Both the called task and the caller should call `CMCloseProfile` when finished with the profile reference.

In your application, you must make sure that `CMCloseProfile` is called once for each time a profile reference is created or cloned. Otherwise, the memory and resources associated with the profile reference may not be properly freed, or an application may attempt to use a profile reference that is no longer valid.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CMApplication.h`

CMCloseProfile

Decrements the reference count for the specified profile reference and, if the reference count reaches 0, frees all private memory and other resources associated with the profile.

```
CMError CMCloseProfile (
   CMProfileRef prof
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) that identifies the profile that may need to be closed.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The ColorSync Manager keeps an internal reference count for each profile reference returned from a call to the `CMOpenProfile` (page 790), `CMNewProfile` (page 788), `CMCopyProfile` (page 740), or `CWNewLinkProfile` (page 830) functions. Calling the function `CMCloneProfileRef` (page 727) increments the count; calling the `CMCloseProfile` function decrements it. The profile remains open as long as the reference count is greater than 0, indicating there is at least one remaining reference to the profile. When the count reaches 0, the ColorSync Manager releases all private memory, files, or resources allocated in association with that profile.

When the ColorSync Manager releases all private memory and resources associated with a profile, any temporary changes your application made to the profile are not saved unless you first call the `CMUpdateProfile` function to update the profile.

When your application passes a copy of a profile reference to an independent task, whether synchronous or asynchronous, it should call the function `CMCloneProfileRef` (page 727) to increment the reference count. Both the called task and the caller should call `CMCloseProfile` when finished with the profile reference.

You call `CMCloneProfileRef` after copying a profile reference, but not after duplicating an entire profile (as with the `CMCopyProfile` function).

When your application passes a copy of a profile reference internally, it may not need to call `CMCloneProfileRef`, as long as the application calls `CMCloseProfile` once for the profile.

In your application, make sure that `CMCloseProfile` is called once for each time a profile reference is created or cloned. Otherwise, the private memory and resources associated with the profile reference may not be properly freed, or an application may attempt to use a profile reference that is no longer valid.

If you create a new profile by calling the `CMNewProfile` function, the profile is saved to disk when you call the `CMCloseProfile` function unless you specified `NULL` as the profile location when you created the profile.

To save changes to a profile before closing it, use the function `CMUpdateProfile` (page 816).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`CMApplication.h`

CMConvertFixedXYZToXYZ

Converts colors specified in XYZ color space whose components are expressed as Fixed XYZ 32-bit signed values of type `CMFixedXYZColor` to equivalent colors expressed as XYZ 16-bit unsigned values of type `CMXYZColor`. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertFixedXYZToXYZ (
   const CMFixedXYZColor *src,
   CMXYZColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of Fixed XYZ colors to convert to XYZ colors.

dst

A pointer to an array containing the list of colors resulting from the conversion specified as XYZ colors.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMConvertFixedXYZToXYZ` function converts one or more colors defined in the Fixed XYZ color space to equivalent colors defined in the XYZ color space. The XYZ color space is device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertFixedXYZToXYZ` function to overwrite the source colors with the resulting converted color specifications.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertHLSToRGB

Converts colors specified in the HLS color space to equivalent colors defined in the RGB color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertHLSToRGB (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of HLS colors to convert to RGB colors.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the RGB color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertHLSToRGB` function converts one or more colors defined in the HLS color space to equivalent colors defined in the RGB color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertHLSToRGB` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertHSVToRGB

Converts colors specified in the HSV color space to equivalent colors defined in the RGB color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertHSVToRGB (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of HSV colors to convert to RGB colors.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the RGB color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertHSVToRGB` function converts one or more colors defined in the HSV color space to equivalent colors defined in the RGB color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertHSVToRGB` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertLabToXYZ

Converts colors specified in the L*a*b* color space to the XYZ color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertLabToXYZ (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src
A pointer to a buffer containing the list of L*a*b* colors to convert to XYZ colors.

white
A pointer to a reference white point.

dst
A pointer to a buffer containing the list of colors as specified in the XYZ color space resulting from the conversion.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertLabToXYZ` function converts one or more colors defined in the L*a*b color space to equivalent colors defined in the XYZ color space. Both color spaces are device independent.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertLuvToXYZ

Converts colors specified in the L*u*v* color space to the XYZ color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertLuvToXYZ (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of L*u*v* colors to convert.

white

A pointer to a reference white point.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the XYZ color space.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMConvertLuvToXYZ` function converts one or more colors defined in the L*u*v color space to equivalent colors defined in the XYZ color space. Both color spaces are device independent.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertRGBToGray

Converts colors specified in the RGB color space to equivalent colors defined in the Gray color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertRGBToGray (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of colors specified in RGB space to convert to colors specified in Gray space.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the Gray color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertRGBToGray` function converts one or more colors defined in the RGB color space to equivalent colors defined in the Gray color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertRGBToGray` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertRGBToHLS

Converts colors specified in the RGB color space to equivalent colors defined in the HLS color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertRGBToHLS (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of RGB colors to convert to HLS colors.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the HLS color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertRGBToHLS` function converts one or more colors defined in the RGB color space to equivalent colors defined in the HLS color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertRGBToHLS` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertRGBToHSV

Converts colors specified in the RGB color space to equivalent colors defined in the HSV color space when the device types are the same. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertRGBToHSV (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src
A pointer to an array containing the list of RGB colors to convert to HSV colors.

dst
A pointer to an array containing the list of colors, resulting from the conversion, as specified in the HSV color space.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertRGBToHSV` function converts one or more colors defined in the RGB color space to equivalent colors defined in the HSV color space. Both color spaces are device dependent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertRGBToHSV` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToFixedXYZ

Converts colors specified in the XYZ color space whose components are expressed as XYZ 16-bit unsigned values of type `CMXYZColor` to equivalent colors expressed as 32-bit signed values of type `CMFixedXYZColor`. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToFixedXYZ (
   const CMXYZColor *src,
   CMFixedXYZColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of XYZ colors to convert to Fixed XYZ colors.

dst

A pointer to an array containing the list of colors resulting from the conversion in which the colors are specified as Fixed XYZ colors.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMConvertXYZToFixedXYZ` function converts one or more colors whose components are defined as XYZ colors to equivalent colors whose components are defined as Fixed XYZ colors. Fixed XYZ colors allow for 32-bit precision. The XYZ color space is device independent.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertXYZToLab

Converts colors specified in the XYZ color space to the L*a*b* color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToLab (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of XYZ colors to convert to L*a*b* colors.

white
A pointer to a reference white point.

dst
A pointer to an array containing the list of L*a*b* colors resulting from the conversion.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertXYZToLab` function converts one or more colors defined in the XYZ color space to equivalent colors defined in the L*a*b* color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertXYZToLab` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToLuv

Converts colors specified in the XYZ color space to the L*u*v* color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToLuv (
   const CMColor *src,
   const CMXYZColor *white,
   CMColor *dst,
   size_t count
);
```

Parameters

src
A pointer to an array containing the list of XYZ colors to convert to L*u*v* colors.

white
A pointer to a reference white point.

dst
A pointer to an array containing the list of colors represented in L*u*v* color space resulting from the conversion.

count
The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertXYZToLuv` function converts one or more colors defined in the XYZ color space to equivalent colors defined in the L*u*v* color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertXYZToLuv` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToXYZ

Converts a source color to a destination color using the specified chromatic adaptation method. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToXYZ (
   const CMColor *src,
   const CMXYZColor *srcIlluminant,
   CMColor *dst,
   const CMXYZColor *dstIlluminant,
   CMChromaticAdaptation method,
   size_t count
);
```

Parameters
src
srcIlluminant
dst
dstIlluminant
method
count

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMConvertXYZToYxy

Converts colors specified in the XYZ color space to the Yxy color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertXYZToYxy (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of XYZ colors to convert to Yxy colors.

dst

A pointer to an array containing the list of colors resulting from the conversion represented in the Yxy color space.

count

The number of colors to convert.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMConvertXYZToYxy` function converts one or more colors defined in the XYZ color space to equivalent colors defined in the Yxy color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertXYZToYxy` function to overwrite the source colors with the resulting converted color specifications.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMConvertYxyToXYZ

Converts colors specified in the Yxy color space to the XYZ color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMConvertYxyToXYZ (
   const CMColor *src,
   CMColor *dst,
   size_t count
);
```

Parameters

src

A pointer to an array containing the list of Yxy colors to convert.

dst

A pointer to an array containing the list of colors, resulting from the conversion, as specified in the XYZ color space.

count

The number of colors to convert.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMConvertYxyToXYZ` function converts one or more colors defined in the Yxy color space to equivalent colors defined in the XYZ color space. Both color spaces are device independent.

If your application does not require that you preserve the source color list, you can pass the pointer to the same color list array as the `src` and `dst` parameters and allow the `CMConvertYxyToXYZ` function to overwrite the source colors with the resulting converted color specifications.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMCopyProfile

Duplicates the specified existing profile.

```
CMError CMCopyProfile (
   CMProfileRef *targetProf,
   const CMProfileLocation *targetLocation,
   CMProfileRef srcProf
);
```

Parameters

targetProf

A pointer to a profile reference of type `CMProfileRef` (page 925). On return, points to the profile copy that was created.

targetLocation

A pointer to a location specification that indicates the location, such as in memory or on disk, where the ColorSync Manager is to create the copy of the profile. A profile is commonly disk-file based. However, to accommodate special requirements, you can create a handle- or pointer-based profile, you can create a profile that is accessed through a procedure provided by your application, or you can create a temporary profile that is not saved after you call the `CMCloseProfile` function. To create a temporary profile, you either specify `cmNoProfileBase` as the kind of profile in the profile location structure or specify `NULL` for this parameter. To specify the location, you use the data type `CMProfileLocation` (page 924).

srcProf

A profile reference to the profile to duplicate.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMCopyProfile` function duplicates an entire open profile whose reference you specify. If you have made temporary changes to the profile, which you have not saved by calling `CMUpdateProfile`, those changes are included in the duplicated profile. They are not saved to the original profile unless you call `CMUpdateProfile` for that profile.

The ColorSync Manager maintains a modified flag to track whether a profile has been modified. After copying a profile, the `CMCopyProfile` function sets the value of the modified flag for that profile to `false`.

Unless you are copying a profile that you created, you should not infringe on copyright protection specified by the profile creator. To obtain the copyright information, you call the function `CMGetProfileElement` (page 768), specifying the `cmCopyrightTag` tag signature for the copyright element (defined in the `CMICCProfile.h` header file).

You should also check the `flags` field of the profile header structure `CM2Header` (page 875) for copyright information. You can test the `cmEmbeddedUseMask` bit of the flags field to determine whether the profile can be used independently. If the bit is set, you should use this profile as an embedded profile only and not copy the profile for your own purposes. The `cmEmbeddedUseMask` mask is described in "Flag Mask Definitions for Version 2.x Profiles" (page 983). The following code snippet shows how you might perform a test using the `cmEmbeddedUseMask` mask:

```
if (myCM2Header.flags & cmEmbeddedUseMask)
{
// profile should only be used as an embedded profile
}
else
{
// profile can be used independently
}
```

A calibration program, for example, might use the `CMCopyProfile` function to copy a device's original profile, then modify the copy to reflect the current state of the device. Or an application might want to copy a profile after unflattening it.

To copy a profile, you must obtain a reference to that profile by either opening the profile or creating it. To open a profile, use the function `CMOpenProfile` (page 790). To create a new profile, use the function `CMNewProfile` (page 788). As an alternative to using the `CMCopyProfile` function to duplicate an entire profile, you can use the same profile reference more than once. To do so, you call the function `CMCloneProfileRef` (page 727) to increment the reference count for the reference each time you reuse it. Calling the `CMCloneProfileRef` function increments the count; calling the function `CMCloseProfile` (page 728) decrements it. The profile remains open as long as the reference count is greater than 0, indicating at least one routine retains a reference to the profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMCopyProfileDescriptionString

Returns the name of a profile as a CFString.

```
CMError CMCopyProfileDescriptionString (
   CMProfileRef prof,
   CFStringRef *str
);
```

Parameters

prof

The profile to query.

str

On ouput, the name of the profile as a CFString.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

If the profile is localized, ColorSync obtains the best localized name for the current process.

Availability

Available in Mac OS X v. 10.3 and later.

Declared In

`CMApplication.h`

CMCopyProfileLocalizedString

Gets one specific string out of a profile

```
CMError CMCopyProfileLocalizedString (
   CMProfileRef prof,
   OSType tag,
   CFStringRef reqLocale,
   CFStringRef *locale,
   CFStringRef *str
);
```

Parameters

prof

The profile to query.

tag

The tag type of profile to query.

reqLocale

The requested locale (optional).

locale

On ouput, points to the locale (optional).

str

On output, points to the dictionary string (optional).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
For example, you pass in the optional tag 'dscm' plus "enUS" for the `reqLocale` parameter, to for a U.S. Enlish string. If a U.S. English string is not found, ColorSync falls back to a reasonable default:

```
err = CMCopyProfileLocalizedString (prof, 'dscm',
                CFSTR("enUS"), nil, &theStr);
```

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMApplication.h`

CMCopyProfileLocalizedStringDictionary

Obtains a CFDictionary which contains the language locale and string for multiple localizations from a given tag.

```
CMError CMCopyProfileLocalizedStringDictionary (
   CMProfileRef prof,
   OSType tag,
   CFDictionaryRef *theDict
);
```

Parameters

prof
The profile to query

tag
The tag type of profile to query

theDict
On output, points to the dictionary .See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
This function allows you to get a CFDictionary which contains the language locale and string for multiple localizations from a given tag.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMApplication.h`

CMCountImageProfiles

Obtains a count of the number of embeded profiles for a given image. (Deprecated in Mac OS X v10.5.)

```
CMError CMCountImageProfiles (
   const FSSpec *spec,
   UInt32 *count
);
```

Parameters

spec

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

count

On output, a count of the embeded profiles for the image

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMCountProfileElements

Counts the number of elements in the specified profile.

```
CMError CMCountProfileElements (
   CMProfileRef prof,
   UInt32 *elementCount
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile to examine.

elementCount

A pointer to an element count. On return, a one-based count of the number of elements.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

Every element in the profile outside the header is counted. A profile may contain tags that are references to other elements. These tags are included in the count.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMCreateProfileIdentifier

Creates a profile identifier for a specified profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMCreateProfileIdentifier (
   CMProfileRef prof,
   CMProfileIdentifierPtr ident,
   UInt32 *size
);
```

Parameters

prof

ident

size

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMDisposeProfileSearch

Frees the private memory allocated for a profile search after your application has completed the search. (Deprecated in Mac OS X v10.5.)

```
void CMDisposeProfileSearch (
   CMProfileSearchRef search
);
```

Parameters

search

A reference to the profile search result list whose private memory is to be released. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 927). See the QuickDraw Reference for a description of the PixMap data type.

Discussion

To set up a search, use the function `CMNewProfileSearch` (page 789). To obtain a reference to a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfile` (page 798). To obtain the file specification for a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfileFileSpec` (page 799). To update the search result list, use the function `CMUpdateProfileSearch` (page 817).

Version Notes

This function is not recommended for use in ColorSync 2.5.

Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 780) for profile searching.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMEmbedImage

Embeds an image with an ICC profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMEmbedImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   CMProfileRef embProf
);
```

Parameters

specFrom
A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto
If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl
A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

embProf
The profile to embed in the image.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMEnableMatchingComment

Inserts a comment into the currently open picture to turn matching on or off. (Deprecated in Mac OS X v10.4.)

```
void CMEnableMatchingComment (
   Boolean enableIt
);
```

Parameters

enableIt

A flag that directs the ColorSync Manager to generate a `cmEnableMatchingPicComment` comment if `true`, or a `cmDisbleMatchingPicComment` comment if `false`.

Discussion

If you call this function when no picture is open, it will have no effect.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

CMEndMatching

Concludes a QuickDraw-specific ColorSync matching session initiated by a previous call to the `NCMBeginMatching` function. (Deprecated in Mac OS X v10.4.)

```
void CMEndMatching (
   CMMatchRef myRef
);
```

Parameters

myRef

A reference to the matching session to end. This reference was previously created and returned by a call to `NCMBeginMatching` function. See the QuickDraw Reference for a description of the PixMap data type.

Discussion

The `CMEndMatching` function releases private memory allocated for the QuickDraw-specific matching session.

After you call the `NCMBeginMatching` function and before you call `CMEndMatching` to end the matching session, embedded color-matching picture comments, such as `cmEnableMatching` and `cmDisableMatching`, are not acknowledged.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

CMFlattenProfile

Transfers a profile stored in an independent disk file to an external profile format that can be embedded in a graphics document. (Deprecated in Mac OS X v10.5.)

```
CMError CMFlattenProfile (
   CMProfileRef prof,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile to flatten.

flags

Reserved for future use.

proc

A pointer to a function that you provide to perform the low-level data transfer. For more information, see the function `CMFlattenProcPtr` (page 855).

refCon

A pointer to a reference constant for application data which the color management module (CMM) passes to the `CMFlattenProcPtr` function each time it calls the function. For example, the reference constant may point to a data structure that holds information required by the `CMFlattenProcPtr` function to perform the data transfer, such as the reference number to a disk file in which the flattened profile is to be stored.

Starting with ColorSync version 2.5, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM.

preferredCMMnotfound

A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM specified by the profile was not available to perform flattening or does not support this function and the default CMM was used. Has the value `false` if the profile's preferred CMM is able to perform flattening.

Starting with ColorSync 2.5, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM. On return, the value of `preferredCMMnotfound` is guaranteed to be `false`.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The ColorSync Manager passes to the CMM the pointer to your profile-flattening function. The CMM calls your function `CMFlattenProcPtr` (page 855) to perform the actual data transfer.

To unflatten a profile embedded in a graphics document to an independent disk file, use the function "Accessing Profile Elements".

Version Notes

Prior to version 2.5, the ColorSync Manager dispatches the `CMFlattenProfile` function to the CMM specified by the profile whose reference you provide. If the preferred CMM is unavailable or it does not support this function, then the default CMM is used.

Starting with ColorSync version 2.5, the ColorSync Manager calls your transfer function directly, without going through the preferred, or any, CMM. As a result, the value returned in the `preferredCMMnotfound` parameter is guaranteed to be `false`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetColorSyncFolderSpec

Obtains the volume reference number and the directory ID for a ColorSync Profiles folder. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetColorSyncFolderSpec (
   short vRefNum,
   Boolean createFolder,
   short *foundVRefNum,
   long *foundDirID
);
```

Parameters

`vRefNum`

The location of the ColorSync profiles folder. In Mac OS X, pass a constant that specifies one of the four possible locations for ColorSync profiles. Pass `kSystemDomain` for profiles located in:

`/System/Library/ColorSync/Profiles`

Pass `kLocalDomain` for profiles located in:

`/Library/ColorSync/Profiles`

Pass `kNetworkDomain` for profiles located in:

`/Network/Library/ColorSync/Profiles`

Pass `kUserDomain` for profiles located in:

`~/Library/ColorSync/Profiles`

In Mac OS 9, pass the reference number of the volume to examine. The volume must be mounted. The constant `kOnSystemDisk` defined in the Folders header file (`Folders.h`) specifies the active system volume.

`createFolder`

A flag you set to `true` to direct the ColorSync Manager to create the ColorSync Profiles folder, if it does not exist. You can use the constants `kCreateFolder` and `kDontCreateFolder`, defined in the `Folders.h` header file, to assign a value to the flag.

`foundVRefNum`

A pointer to a volume reference number. On return, the volume reference number for the volume on which the ColorSync Profiles folder resides.

foundDirID
A pointer to a directory ID. On return, the directory ID for the volume on which the ColorSync Profiles folder resides.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
If the ColorSync Profiles folder does not already exist, you can use this function to create it.

Version Notes
Starting with version 2.5, the name and location of the profile folder changed.

Your application should use the function `CMIterateColorSyncFolder` (page 780), available starting in ColorSync version 2.5, or one of the search functions described in "Searching for Profiles Prior to ColorSync 2.5", to search for a profile file, even if it is only looking for one file. Do not search for a profile file by obtaining the location of the profiles folder and searching for the file directly.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetColorSyncVersion

Gets ColorSync version information.

```
CMError CMGetColorSyncVersion (
   UInt32 *version
);
```

Parameters

version
On output, points to the version of ColorSync installed on the system.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
`CMGetColorSyncVersion` relieves you from having to call `Gestalt` to find out the version of ColorSync installed on the system.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetCWInfo

Obtains information about the color management modules (CMMs) used for a specific color world. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetCWInfo (
   CMWorldRef cw,
   CMCWInfoRecord *info
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 942) about which you want information.

The functions `NCWNewColorWorld` (page 846) and `CWConcatColorWorld` (page 823) both allocate color world references of type `CMWorldRef` (page 942).

info

A pointer to a color world information record of type `CMCWInfoRecord` (page 888) that your application supplies. On return, the ColorSync Manager returns information in this structure describing the number of CMMs involved in the matching session and the CMM type and version of each CMM used.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

This discussion is accurate for versions of ColorSync prior to 2.5. See the version notes below for changes starting with version 2.5.

To learn whether one or two CMMs are used for color matching and color checking in a given color world and to obtain the CMM type and version number of each CMM used, your application must first obtain a reference to the color world. To obtain a reference to a ColorSync color world, you (or some other process) must have created the color world using the function `NCWNewColorWorld` (page 846) or the function `CWConcatColorWorld` (page 823).

The source and destination profiles you specify when you create a color world identify their preferred CMMs, and you explicitly identify the profile whose CMM is used for a device link profile or a concatenated color world. However, you cannot be certain if the specified CMM will actually be used until the ColorSync Manager determines internally if the CMM is available and able to perform the requested function. For example, when the specified CMM is not available, the default CMM is used.

The `CMGetCWInfo` function identifies the CMM or CMMs to use. Your application must allocate a data structure of type `CMCWInfoRecord` and pass a pointer to it in the `info` parameter. The `CMGetCWInfo` function returns the color world information in this structure. The structure includes a `cmmCount` field identifying the number of CMMs that will be used and an array of two members containing structures of type `CMMInfoRecord` (page 909). The `CMGetCWInfo` function returns information in one or both of the CMM information records depending on whether one or two CMMs are used.

Version Notes

Starting with ColorSync 2.5, a user can select a preferred CMM with the ColorSync control panel. If the user has selected a preferred CMM, and if it is available, then it will be used for all color conversion and matching operations.

Availability

Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetDefaultDevice

Gets the default device.

```
CMError CMGetDefaultDevice (
   CMDeviceClass deviceClass,
   CMDeviceID *deviceID
);
```

Parameters

deviceClass

The device class whose default device you want to get. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

On return, points to the device ID for the default device.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
For each class of device, a device management layer may establish which of the registered devices is the default. This helps keep color management choices to a minimum and allows for some automatic features to be enabled, such as the "Default printer" as an output profile selection.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetDefaultProfileBySpace

Gets the default profile for the specified color space.

```
CMError CMGetDefaultProfileBySpace (
   OSType dataColorSpace,
   CMProfileRef *prof
);
```

Parameters

dataColorSpace

A four-character identifier of type `OSType`. You pass a color space signature that identifies the color space you wish to get the default profile for. The currently-supported values are `cmRGBData`, `cmCMYKData`, `cmLabData`, and `cmXYZData`. These constants are a subset of the constants described in "Color Space Signatures" (page 969). If you supply a value that is not supported, the `CMGetDefaultProfileBySpace` function returns an error value of `paramErr`.

prof

A pointer to a profile reference. On return, the reference specifies the current profile for the color space specified by `dataColorSpace`. `CMGetDefaultProfileBySpace` currently supports only file-based profiles.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMGetDefaultProfileBySpace` function currently supports the RGB, CMYK, Lab, and XYZ color spaces. The signature constants for these color spaces (shown above with the `dataColorSpace` parameter description) are described in "Color Space Signatures" (page 969). Support for additional color spaces may be provided in the future. `CMGetDefaultProfileBySpace` returns an error value of `paramErr` if you pass a color space constant it does not currently support.

The `CMGetDefaultProfileBySpace` function always attempts to return a file-based profile for a supported color space. For example, if the user has not specified a default profile in the ColorSync control panel for the specified color space, or if the profile is not found (the user may have deleted the profiles in the ColorSync Profiles folder or even the folder itself), `CMGetDefaultProfileBySpace` creates a profile, stores it on disk, and returns a reference to that profile. However, you should always check for an error return—for example, a user may have booted from a CD, so that `CMGetDefaultProfileBySpace` cannot save a profile file to disk.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetDefaultProfileByUse

Obtains the users' preferred device profile setting.

```
CMError CMGetDefaultProfileByUse (
   OSType use,
   CMProfileRef *prof
);
```

Parameters

use

A value that specifies the device type for which to obtain the profile.

prof

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetDeviceDefaultProfileID

Gets the default profile ID for a given device.

```
CMError CMGetDeviceDefaultProfileID (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID *defaultProfID
);
```

Parameters

deviceClass

The device class to query. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

The device ID to query.

defaultID

On output, points to the id of the default profile for this device.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

Device drivers and host software can set the default profile for a given device using the function `CMSetDeviceDefaultProfileID`.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMGetDeviceFactoryProfiles

Retrieves the original profiles for a given device.

```
CMError CMGetDeviceFactoryProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID *defaultProfID,
   UInt32 *arraySize,
   CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass

The device class to query. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

The device ID to query.

defaultProfID

A pointer to the default profile for this device.

arraySize

A pointer to the size of the array to be returned. You can first call this routine to get the size returned, then call it again with the size of the buffer to receive the array.

deviceProfiles

On output, points to the profile array. You can first pass `NULL` in this parameter to receive the size of the array in the `arraySize` parameter. Then, once the appropriate amount of storage has been allocated, a pointer to it can be passed in this parameter to have the array copied to that storage.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
This function allows you to retrieve the original profiles for a given device. These may differ from the actual profiles in use for that device, in the case where any factory profiles have been replaced (updated). To get the actual profiles in use, call `CMGetDeviceProfiles`.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetDeviceInfo

Gets information about a specified device.

```
CMError CMGetDeviceInfo (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceInfo *deviceInfo
);
```

Parameters

deviceClass

A device class to query. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

A device ID to query. You can pass `cmDefaultDeviceID`.

deviceInfo

On input, points to a device information dictionary On output, the dictionary is filled with device information. If, on input, `deviceInfo->deviceName` is `nil` then the name is not returned. If you wants the device name dictionary returned, you should provide in `deviceInfo->deviceName` the address where this routine should store the CFDictionaryRef. The caller is responsible for disposing of the name dictionary.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
CMDeviceIntegration.h

CMGetDeviceProfile

Gets a profile used by a given device.

```
CMError CMGetDeviceProfile (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID profileID,
   CMProfileLocation *profileLoc
);
```

Parameters

deviceClass
The device class for the device whose profile you want to get. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID
The device ID for the device whose profile you want to get.

defaultID
The ID of the default profile for this device.

deviceProfLoc
On return, the location of the profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
CMDeviceIntegration.h

CMGetDeviceProfiles

Gets the profiles used by a given device. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetDeviceProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   UInt32 *arraySize,
   CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass
The device class for the device whose profiles you want to get. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID
The device ID for the device whose profiles you want to get.

arraySize
A pointer to the size of the array to be returned. You can first call this routine to get the size returned, then call it again with the size of the buffer to receive the array.

deviceProfiles
On output, an array of profiles used by the device. You can first pass `NULL` in this parameter to receive the size of the array in the `arraySize` parameter. Then, once the appropriate amount of storage has been allocated, a pointer to it can be passed in this parameter to have the array copied to that storage.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMDeviceIntegration.h`

CMGetDeviceState

Gets the state of a device.

```
CMError CMGetDeviceState (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceState *deviceState
);
```

Parameters
deviceClass
A device class to query. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID
A device ID to query. You can pass `cmDefaultDeviceID`.

deviceState
On output, points to the device state. See "Device States" (page 980) for the values that can be returned.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMGetGammaByAVID

Obtains the gamma value for the specified display device.

```
CMError CMGetGammaByAVID (
   CMDisplayIDType theID,
   CMVideoCardGamma *gamma,
   UInt32 *size
);
```

Parameters

theID

A Display Manager ID value. You pass the ID value for the display device for which to set the gamma.

gamma

size

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetImageSpace

Returns the signature of the data color space in which the color values of colors in an image are expressed. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetImageSpace (
   const FSSpec *spec,
   OSType *space
);
```

Parameters

spec

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

space

The signature of the data color space of the color values of colors for the image file is returned here.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMGetIndImageProfile

Obtains a specific embeded profile for a given image. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetIndImageProfile (
   const FSSpec *spec,
   UInt32 index,
   CMProfileRef *prof
);
```

Parameters

spec

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

index

The numeric index of the profile to return.

prof

On output, points to the profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMGetIndNamedColorValue

Obtains device and PCS color values for a specific named color index from a named color space profile.

```
CMError CMGetIndNamedColorValue (
   CMProfileRef prof,
   UInt32 index,
   CMColor *deviceColor,
   CMColor *PCSColor
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to a named color space profile to obtain color values from.

index

A one-based index value for a named color.

deviceColor

A pointer to a device color. On return, a device color value in `CMColor` union format. If the profile does not contain device values, `deviceColor` is undefined.

PCSColor

A pointer to a profile connection space color. On return, an interchange color value in `CMColor` union format.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Based on the passed named color index, the `CMGetIndNamedColorValue` function does a lookup into the named color tag and returns device and PCS values. If the index is greater than the number of named colors, `CMGetIndNamedColorValue` returns an error code.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetIndProfileElement

Obtains the element data corresponding to a particular index from the specified profile.

```
CMError CMGetIndProfileElement (
   CMProfileRef prof,
   UInt32 index,
   UInt32 *elementSize,
   void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile containing the element.

index

The index of the element whose data you want to obtain. This is a one-based element index within the range returned as the `elementCount` parameter of the `CMCountProfileElements` function.

elementSize

A pointer to an element data size. On input, specify the size of the element data to copy (except when `elementData` is set to `NULL`). Specify `NULL` to copy the entire element data. To obtain a portion of the element data, specify the number of bytes to be copy.

On return, the size of the element data actually copied.

elementData

A pointer to memory for element data. On input, you allocate memory. On return, this buffer holds the element data.

To obtain the element size in the `elementSize` parameter without copying the element data to this buffer, specify `NULL` for this parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Before you call the `CMGetIndProfileElement` function to obtain the element data for an element at a specific index, you first determine the size in bytes of the element data. To determine the data size, you can

- call the function `CMGetIndProfileElementInfo` (page 761), passing the element's index
- call the `CMGetIndProfileElement` function itself, specifying a pointer to an unsigned long data type in the `elementSize` field and a `NULL` value in the `elementData` field

Once you have determined the size of the element data, you allocate a buffer to hold as much of the data as you need. If you want all of the element data, you specify `NULL` in the `elementSize` parameter. If you want only a portion of the element data, you specify the number of bytes you want in the `elementSize` parameter. You supply a pointer to the data buffer in the `elementData` parameter. After calling `CMGetIndProfileElement`, the `elementSize` parameter contains the size in bytes of the element data actually copied.

Before calling this function, you should call the function `CMCountProfileElements` (page 744). It returns the profile's total element count in the `elementCount` parameter.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetIndProfileElementInfo

Obtains the element tag and data size of an element by index from the specified profile.

```
CMError CMGetIndProfileElementInfo (
   CMProfileRef prof,
   UInt32 index,
   OSType *tag,
   UInt32 *elementSize,
   Boolean *refs
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 925) to the profile containing the element.

index
A one-based element index within the range returned by the `elementCount` parameter of the `CMCountProfileElements` function.

tag
A pointer to an element signature. On return, the tag signature of the element corresponding to the index.

elementSize
A pointer to an element size. On return, the size in bytes of the element data corresponding to the tag.

refs
A pointer to a reference count flag. On return, set to `true` if more than one tag in the profile refers to element data associated with the tag corresponding to the index.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The index order of elements is determined internally by the ColorSync Manager and is not publicly defined.

Before calling the `CMGetIndProfileElementInfo` function, you should call the function `CMCountProfileElements` (page 744), which returns the total number of elements in the profile in the `elementCount` parameter. The number you specify for the `index` parameter when calling `CMGetIndProfileElementInfo` should be in the range of 1 to `elementCount`; otherwise the function will return a result code of `cmIndexRangeErr`.

You might want to call this function, for example, to print out the contents of a profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetNamedColorIndex

Obtains a named color index for a specific color name from a named color space profile.

```
CMError CMGetNamedColorIndex (
   CMProfileRef prof,
   StringPtr name,
   UInt32 *index
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to a named color space profile to obtain a named color index from.

name

A pointer to a Pascal string. You supply a color name string value for the color to obtain the color index for.

index

A pointer to an index value. On return, an index value for a named color.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Based on the passed color name, the `CMGetNamedColorIndex` function does a lookup into the named color tag and, if the name is found in the tag, returns the index. Otherwise, `CMGetNamedColorIndex` returns an error code.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetNamedColorInfo

Obtains information about a named color space from its profile reference.

```
CMError CMGetNamedColorInfo (
   CMProfileRef prof,
   UInt32 *deviceChannels,
   OSType *deviceColorSpace,
   OSType *PCSColorSpace,
   UInt32 *count,
   StringPtr prefix,
   StringPtr suffix
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to a named color space profile to obtain named color information from.

deviceChannels

A pointer to a count value. On return, the number of device channels in the color space for the profile. It should agree with the "data color space" field in the profile header. For example, Pantone maps to CMYK, a four-channel color space. A value of 0 indicates no device channels were available.

deviceColorSpace

A pointer to a device color space. On return, a device color space, such as CMYK.

PCSColorSpace

A pointer to a profile connection space color space. On return, an interchange color space, such as Lab.

count

A pointer to a count value. On return, the number of named colors in the profile.

prefix

A pointer to a Pascal string. On return, the string contains a prefix, such as "Pantone", for each color name. The prefix identifies the named color system described by the profile.

suffix

A pointer to a Pascal string. On return, the string contains a suffix for each color name, such as "CVC".

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMGetNamedColorInfo` function returns information about the named color space referred to by the passed profile reference.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetNamedColorName

Obtains a named color name for a specific named color index from a named color space profile.

```
CMError CMGetNamedColorName (
   CMProfileRef prof,
   UInt32 index,
   StringPtr name
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to a named color space profile to obtain a named color name from.

index

An index value for a named color to obtain the color name for.

name

A pointer to a Pascal string. On return, a color name string.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

Based on the passed color name index, the `CMGetNamedColorName` function does a lookup into the named color tag and returns the name. If the index is greater than the number of named colors, `CMGetNamedColorName` returns an error code.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetNamedColorValue

Obtains device and PCS color values for a specific color name from a named color space profile.

```
CMError CMGetNamedColorValue (
   CMProfileRef prof,
   StringPtr name,
   CMColor *deviceColor,
   CMColor *PCSColor
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to a named color space profile to obtain color values from.

name

A pointer to a Pascal string. You supply a color name string for the color to get information for.

deviceColor

A pointer to a device color. On return, a device color value in `CMColor` union format. If the profile does not contain device values, `deviceColor` is undefined.

PCSColor

A pointer to a profile connection space color. On return, an interchange color value in `CMColor` union format.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Based on the passed color name, the `CMGetNamedColorValue` function does a lookup into the named color tag and, if the name is found in the tag, returns device and color PCS values. Otherwise, `CMGetNamedColorValue` returns an error code.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPartialProfileElement

Obtains a portion of the element data from the specified profile based on the specified element tag signature.

```
CMError CMGetPartialProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 offset,
   UInt32 *byteCount,
   void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile containing the target element.

tag

The tag signature for the element in question. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

offset

Beginning from the first byte of the element data, the offset from which to begin copying the element data.

byteCount

A pointer to a data byte count. On input, the number of bytes of element data to copy, beginning from the offset specified by the `offset` parameter. On return, the number of bytes actually copied.

elementData

A pointer to memory for element data. On input, you pass a pointer to allocated memory. On return, this buffer holds the element data.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMGetPartialProfileElement` function allows you to copy any portion of the element data beginning from any offset into the data. For the `CMGetPartialProfileElement` function to copy the element data and return it to you, your application must allocate a buffer in memory to hold the data.

You cannot use this function to obtain a portion of the `CM2Header` profile header. Instead, you must call the function `CMGetProfileHeader` (page 769) to get the entire profile header and read its contents.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPreferredCMM

Identifies the preferred CMM specified by the ColorSync control panel. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetPreferredCMM (
   OSType *cmmType,
   Boolean *prefCMMnotfound
);
```

Parameters

cmmType
A pointer to an `OSType`. On return, the component subtype for the preferred CMM. For example, the subtype for ColorSync's default CMM is `'appl'` and the subtype for the Kodak CMM is `'KCMS'`. A return value of `NULL` indicates the preferred CMM in the ColorSync control panel is set to Automatic.

preferredCMMnotfound
A pointer to a Boolean flag for whether the preferred CMM was not found. On return, has the value `true` if the CMM was not found, `false` if it was found.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMGetPreferredCMM` function returns in the `cmmType` parameter a value that identifies the preferred CMM the user last specified in the ColorSync control panel. `CMGetPreferredCMM` returns `false` in the `preferredCMMnotfound` parameter if the preferred CMM is currently available and `true` if it is not. The preferred CMM may not be available, for example, because a user specifies a preferred CMM in the ColorSync control panel, then reboots with extensions off. ColorSync does not change the preferred CMM setting when the preferred CMM is not available.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetProfileByAVID

Gets the current profile for a monitor.

```
CMError CMGetProfileByAVID (
   CMDisplayIDType theID,
   CMProfileRef *prof
);
```

Parameters

theAVID

A Display Manager ID value. You pass the ID value for the monitor for which to get the profile.

prof

A pointer to a profile reference. On return, a reference to the current profile for the monitor specified by `theAVID`.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

If the Display Manager supports ColorSync, the `CMGetProfileByAVID` function calls on the Display Manager to get the profile for the specified display. This is the case if the version of the Display Manager is 2.2.5 or higher (if `gestaltDisplayMgrAttr` has the `gestaltDisplayMgrColorSyncAware` bit set).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetProfileDescriptions

Obtains the description tag data for a specified profile.

```
CMError CMGetProfileDescriptions (
   CMProfileRef prof,
   char *aName,
   UInt32 *aCount,
   Str255 mName,
   ScriptCode *mCode,
   UniChar *uName,
   UniCharCount *uCount
);
```

Parameters

prof

A reference to the profile from which to obtain the description info.

aName

On output, a pointer to the profile name as a 7-bit Roman ASCII string.

aCount

On output, a pointer to a count of the number of characters returned in the `aName` field.

mName

On output, a pointer to the localized profile name string in Mac script-code format.

mCode

On output, a pointer the script code corresponding to the name string returned in the `mName` parameter.

uName

On output, a pointer to localizedUnicode profile name string.

uCount

On output, a pointer to a count of the number of Unicode (2-byte) characters returned in the `uName` parameter.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

Use this function to get the description tag data for a given profile. The ICC Profile Format Specification (available at http://www.color.org) includes a description tag (`'desc'`), designed to provide more information about a profile than can be contained in a file name. This is especially critical on file systems with 8.3 names. The tag data can consist of up to three separate pieces (strings) of information for a profile. These different strings are designed to allow for display in different languages or on different computer systems. Applications typically use one of the strings to show profiles in a list or a pop-up menu.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetProfileElement

Obtains element data from the specified profile based on the specified element tag signature.

```
CMError CMGetProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 *elementSize,
   void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile containing the target element.

tag

The tag signature (for example, 'A2B0', or constant `cmAToB0Tag`) for the element in question. The tag identifies the element. For a complete list of the public tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

elementSize

A pointer to a size value. On input, you specify the size of the element data to copy. Specify `NULL` to copy the entire element data. To obtain a portion of the element data, specify the number of bytes to copy.

On return, the size of the data returned.

elementData

A pointer to memory for element data. On input, you allocate memory. On return, this buffer holds the element data.

To obtain the element size in the `elementSize` parameter without copying the element data to this buffer, specify `NULL` for this parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Before you call the `CMGetProfileElement` function to obtain the element data for a specific element, you must know the size in bytes of the element data so you can allocate a buffer to hold the returned data.

The `CMGetProfileElement` function serves two purposes: to get an element's size and to obtain an element's data. In both instances, you provide a reference to the profile containing the element in the `prof` parameter and the tag signature of the element in the `tag` parameter.

To obtain the element data size, call the `CMGetProfileElement` function specifying a pointer to an unsigned long data type in the `elementSize` field and a `NULL` value in the `elementData` field.

After you obtain the element size, you should allocate a buffer large enough to hold the returned element data, then call the `CMGetProfileElement` function again, specifying `NULL` in the `elementSize` parameter to copy the entire element data and a pointer to the data buffer in the `elementData` parameter.

To copy only a portion of the element data beginning from the first byte, allocate a buffer the size of the number of bytes of element data you want to obtain and specify the number of bytes to copy in the `elementSize` parameter. In this case, On return the `elementSize` parameter contains the size in bytes of the element data actually returned.

You cannot use the `CMGetProfileElement` function to copy a portion of element data beginning from an offset into the data. To copy a portion of the element data beginning from any offset, use the function `CMGetPartialProfileElement` (page 765).

You cannot use this function to obtain a portion of the `CM2Header` profile header. Instead, you must call the function `CMGetProfileHeader` (page 769) to copy the entire profile header and read its contents.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetProfileHeader

Obtains the profile header for the specified profile.

```
CMError CMGetProfileHeader (
   CMProfileRef prof,
   CMAppleProfileHeader *header
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile whose header is to be copied.

header

A pointer to a profile header. On input, depending on the profile version, you may allocate a ColorSync 2.x or 1.0 header. On return, contains the profile data. For information about the ColorSync 2.x profile header structure, see `CM2Header` (page 875). For information about the ColorSync 1.0 header, see `CMHeader` (page 898).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMGetProfileHeader` function returns the header for a ColorSync 2.x or ColorSync 1.0 profile. To return the header, the function uses a union of type `CMAppleProfileHeader` (page 881), with variants for version 1.0 and 2.x headers.

A 32-bit version value is located at the same offset in either header. For ColorSync 2.x profiles, this is the `profileVersion` field. For ColorSync 1.0 profiles, this is the `applProfileVersion` field. You can inspect the value at this offset to determine the profile version, and interpret the remaining header fields accordingly.

To copy a profile header to a profile after you modify the header's contents, use the function `CMSetProfileHeader` (page 813).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMGetProfileLocation

Obtains the location of a profile based on the specified profile reference. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetProfileLocation (
   CMProfileRef prof,
   CMProfileLocation *location
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925). Before calling `CMGetProfileLocation`, you set the reference to specify the profile you wish to obtain the location for.

theProfile

A pointer to a profile location structure of type `CMProfileLocation` (page 924). On return, specifies the location of the profile. Commonly, a profile is disk-file based, but it may instead be temporary, handle-based, pointer-based, or accessed through a procedure supplied by your application.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
When your application calls the `CMValidateProfile` function, the ColorSync Manager dispatches the function to the CMM specified by the `CMMType` header field of the profile whose reference you specify. The preferred CMM can support this function or not.

To open a profile and obtain a reference to it, use the function `CMOpenProfile` (page 790).

Version Notes
This function is not recommended for use in ColorSync 2.5.

Starting with ColorSync version 2.5, you should use the function `NCMGetProfileLocation` (page 841) instead of `CMGetProfileLocation`.

As of version 2.5, if you call `CMGetProfileLocation`, it will just call `NCMGetProfileLocation` in turn, passing the profile specified by `prof`, the profile location specified by `theProfile`, and a location size value of `cmOriginalProfileLocationSize`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetProfileMD5

Gets the MD5 checksum from the profile header (message digest)

```
CMError CMGetProfileMD5 (
   CMProfileRef prof,
   CMProfileMD5 digest
);
```

Parameters
prof
digest

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
You can call this function to determine if two profiles are identical, or if a profile has changed over time. You can access this new MD5 checksum directly in the profile header, or use the function `CMGetProfileMD5`. This function has the advantage that it works with both ICC 4 profiles and earlier profiles.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
CMApplication.h

CMGetProfileRefCount

Obtains the current reference count for the specified profile.

```
CMError CMGetProfileRefCount (
   CMProfileRef prof,
   long *count
);
```

Parameters

prof

A profile reference of type CMProfileRef (page 925) to the profile whose reference count is obtained.

count

A pointer to a reference count. On return, the reference count for the specified profile reference.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The ColorSync Manager keeps an internal reference count for each profile reference returned from calls such as CMOpenProfile (page 790) or CMNewProfile (page 788). Calling the function CMCloneProfileRef (page 727) increments the count; calling the function CMCloseProfile (page 728) decrements it. The profile remains open as long as the reference count is greater than 0, indicating at least one routine retains a reference to the profile. When the count reaches 0, the ColorSync Manager releases all memory, files, or resources allocated in association with that profile.

An application that manages profiles closely can call the CMGetProfileRefCount function to obtain the reference count for a profile reference, then perform special handling if necessary, based on the reference count.

To copy a profile with the function CMCopyProfile (page 740), you must obtain a reference to that profile by either opening the profile or creating it. To open a profile, use the function CMOpenProfile (page 790). To create a new profile, use the function CMNewProfile (page 788). As an alternative to using the CMCopyProfile function to duplicate an entire profile, you can use the same profile reference more than once. To do so, you call the function CMCloneProfileRef (page 727) to increment the reference count for the reference each time you reuse it. Calling the CMCloneProfileRef function increments the count; calling the function CMCloseProfile (page 728) decrements it. The profile remains open as long as the reference count is greater than 0, indicating at least one routine retains a reference to the profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CMGetPS2ColorRendering

Obtains the color rendering dictionary (CRD) element data usable as the parameter to the PostScript `setColorRendering` operator, which specifies the PostScript color rendering dictionary to use for the following graphics data.

```
CMError CMGetPS2ColorRendering (
   CMProfileRef srcProf,
   CMProfileRef dstProf,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf

A profile reference to a profile that supplies the rendering intent for the CRD.

dstProf

A profile reference to a profile from which to extract the CRD data.

flags

If the value of `flags` is equal to `cmPS8bit`, the generated PostScript will utilize 8-bit encoding whenever possible to achieve higher data compaction. If the value of `flags` is not equal to `cmPS8bit`, the generated data will be 7-bit safe, in either ASCII or ASCII base-85 encoding.

proc

A pointer to a callback flatten function to perform the data transfer. For information, see the function `CMFlattenProcPtr` (page 855).

refCon

An untyped pointer to arbitrary data supplied by your application. `CMGetPS2ColorSpace` passes this data in calls to your `CMFlattenProcPtr` (page 855) function.

preferredCMMnotfound

A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value `false`.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMGetPS2ColorRendering` function obtains CRD data from the profile specified by the `dstProf` parameter. To be valid, the parameter must specify an output profile with at most four components. The CMM uses the rendering intent from the profile specified by the `srcProf` parameter to determine which of the PostScript tags (`ps2CR0Tag`, `ps2CR1Tag`, `ps2CR2Tag`, or `ps2CR3Tag`) to use in creating the CRD. If none of these tags exists in the profile, the CMM creates the CRD from one of the multidimensional table tags (`cmAToB0`, `cmAToB1`, or `cmAToB2`), again chosen according to the rendering intent of the profile specified by the `srcProf` parameter.

This function is dispatched to the CMM component specified by the destination profile. If the designated CMM is not available or the CMM does not implement this function, the ColorSync Manager dispatches this function to the default CMM.

The CMM obtains the PostScript data and passes it to your low-level data transfer procedure, specified by the `proc` parameter. The CMM converts the data into a PostScript stream and calls your procedure as many times as necessary to transfer the data to it. Typically, the low-level data transfer function returns this data to the calling application or device driver to pass to a PostScript printer.

Before your application or device driver sends the CRD to the printer, it can call the function `CMGetPS2ColorRenderingVMSize` (page 775) to determine the virtual memory size of the CRD.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPS2ColorRenderingIntent

Obtains the rendering intent element data in text format usable as the parameter to the PostScript `findRenderingIntent` operator, which specifies the color-matching option for subsequent graphics data.

```
CMError CMGetPS2ColorRenderingIntent (
   CMProfileRef srcProf,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf
A profile reference to the source profile that defines the data color space and identifies the preferred CMM.

flags
If the value of `flags` is equal to `cmPS8bit`, the generated PostScript will utilize 8-bit encoding whenever possible to achieve higher data compaction. If the value of `flags` is not equal to `cmPS8bit`, the generated data will be 7-bit safe, in either ASCII or ASCII base-85 encoding.

proc
A low-level data transfer function supplied by the calling application to receive the PostScript data from the CMM. For more information, see the function `CMFlattenProcPtr` (page 855).

refCon
An untyped pointer to arbitrary data supplied by your application. `CMGetPS2ColorSpace` passes this data in calls to your `CMFlattenProcPtr` (page 855) function.

preferredCMMnotfound
A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value `false`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMGetPS2ColorRenderingIntent` function obtains PostScript rendering intent information from the header of the source profile. It returns data by calling your low-level data transfer procedure and passing the PostScript data to it. Typically, your low-level data transfer function returns this data to the calling application or device driver to pass to a PostScript printer.

The `CMGetPS2ColorRenderingIntent` function is dispatched to the CMM component specified by the source profile. If the designated CMM is not available or the CMM does not implement this function, then ColorSync dispatches the function to the default CMM.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPS2ColorRenderingVMSize

Determines the virtual memory size of the color rendering dictionary (CRD) for a printer profile before your application or driver obtains the CRD and sends it to the printer.

```
CMError CMGetPS2ColorRenderingVMSize (
   CMProfileRef srcProf,
   CMProfileRef dstProf,
   UInt32 *vmSize,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf
A profile reference to a profile that supplies the rendering intent for the CRD.

dstProf
A profile reference to the destination printer profile.

vmSize
A pointer to a memory size. On return, the virtual memory size of the CRD.

preferredCMMnotfound
A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value `false`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Your application or device driver can call this function to determine if the virtual memory size of the color rendering dictionary exceeds the printer's capacity before sending the CRD to the printer. If the printer's profile contains the Apple-defined optional tag `'psvm'` described in `CMConcatProfileSet` (page 887), then the default CMM will return the data supplied by this tag specifying the CRD virtual memory size for the rendering intent's CRD. If the printer's profile does not contain this tag, then the CMM uses an algorithm to assess the VM size of the CRD, in which case the assessment can be larger than the actual maximum VM size.

The CMM uses the profile specified by the `srcProf` parameter to determine the rendering intent to use.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetPS2ColorSpace

Obtains color space element data in text format usable as the parameter to the PostScript `setColorSpace` operator, which characterizes the color space of subsequent graphics data.

```
CMError CMGetPS2ColorSpace (
   CMProfileRef srcProf,
   UInt32 flags,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

srcProf
A profile reference to the source profile that defines the data color space and identifies the preferred CMM.

flags
If the value of `flags` is equal to `cmPS8bit`, the generated PostScript will utilize 8-bit encoding whenever possible to achieve higher data compaction. If the value of `flags` is not equal to `cmPS8bit`, the generated data will be 7-bit safe, in either ASCII or ASCII base-85 encoding.

proc
A pointer to a callback flatten function to receive the PostScript data from the CMM. For information, see the function `CMFlattenProcPtr` (page 855).

refCon
An untyped pointer to arbitrary data supplied by your application. `CMGetPS2ColorSpace` passes this data in calls to your `CMFlattenProcPtr` (page 855) function.

preferredCMMnotfound
A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM corresponding to profile was not available or if it was unable to perform the function and the default CMM was used. Otherwise, has the value `false`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMGetPS2ColorSpace` function obtains PostScript color space data from the source profile. The valid profile classes for the `CMGetPS2ColorSpace` function are display, input, and output profiles with at most four components.

To determine which profile elements to use to generate the PostScript color space data, the CMM:

- uses the PostScript `cmPS2CSATag`, if it exists
- otherwise, uses the multidimensional table tag (`cmAToB0`, `cmAToB1`, or `cmAToB2`), if it exists, for the rendering intent currently specified by the profile

- otherwise, uses the multidimensional table tag `cmAToB0`, if it exists
- otherwise, for display profiles only, uses the tristimulus tags (`cmRedColorantTag`, `cmGreenColorantTag`, `cmBlueColorantTag`) and the tonal curve tags (`cmRedTRCTag`, `cmGreenTRCTag`, and `cmBlueTRCTag`)

The CMM obtains the PostScript data from the profile and calls your low-level data transfer procedure passing the PostScript data to it. The CMM converts the data into a PostScript stream and calls your procedure as many times as necessary to transfer the data to it.

Typically, the low-level data transfer function returns this data to the calling application or device driver to pass to a PostScript printer as an operand to the PostScript `setcolorspace` operator, which defines the color space of graphics data to follow.

The `CMGetPS2ColorSpace` function is dispatched to the CMM component specified by the source profile. If the designated CMM is not available or the CMM does not implement this function, then the ColorSync Manager dispatches the function to the default CMM.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMGetScriptProfileDescription

Obtains the internal name (or description) of a profile and the script code identifying the language in which the profile name is specified from the specified profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMGetScriptProfileDescription (
   CMProfileRef prof,
   Str255 name,
   ScriptCode *code
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 925) to the profile whose profile name and script code are obtained.

name
A pointer to a name string. On return, the profile name.

code
A pointer to a script code. On return, the script code.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The element data of the text description tag (which has the signature `'desc'` or constant `cmSigProfileDescriptionType`, defined in the `CMICCProfile.h` header file) specifies the profile name and script code. The `name` parameter returns the profile name as a Pascal string. Use this function so that your application does not need to obtain and parse the element data, which contains other information.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMGetSystemProfile

Obtains a reference to the current system profile.

```
CMError CMGetSystemProfile (
   CMProfileRef *prof
);
```

Parameters

prof

A pointer to a profile reference of type `CMProfileRef` (page 925). On return, a reference to the current system profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The following functions allow you to pass `NULL` as a parameter value to specify the system profile as a source or destination profile:

- `CMNewProfile` (page 788)
- `NCWNewColorWorld` (page 846)
- `NCMBeginMatching` (page 838)
- `NCMDrawMatchedPicture` (page 840)

Note that instead of passing `NULL`, you can pass a profile reference to a specific profile, including the system profile.

If you want to specify the system profile for any other function that requires a profile reference, such as `CWConcatColorWorld` (page 823) and `CWNewLinkProfile` (page 830), you must use an explicit reference. You can obtain such a reference with the `CMGetSystemProfile` function.

There are other reasons you might need to obtain a reference to the current system profile. For example, your application might need to display the name of the current system profile to a user.

To identify the location of the physical file, call the function `CMGetProfileLocation` (page 770).

When your application has finished using the current system profile, it must close the reference to the profile by calling the function `CMCloseProfile` (page 728).

Version Notes
Starting with version 2.5, use of the system profile has changed. So rather than call `CMGetSystemProfile` to obtain a reference to the system profile, you may be able to obtain a profile that is more appropriate for the current operation by calling `CMGetDefaultProfileBySpace` (page 752) to get the default profile for a color space or by calling `CMGetProfileByAVID` (page 767) to get the profile for a specific display.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMIterateCMMInfo

Iterates through the color management modules installed on the system.

```
CMError CMIterateCMMInfo (
   CMMIterateUPP proc,
   UInt32 *count,
   void *refCon
);
```

Parameters

proc
A calling-program-supplied callback function that allows your application to monitor progress or abort the operation.

count
A pointer to the number of available CMMs.

refCon
A reference constant containing data specified by the calling application program.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMIterateCMMInfo` function returns information for all CMMs installed on the system. The caller can pass nil for the `CMMIterateUPP` param to simply get a count of CMMs. If a `CMMIterateUPP` proc is provided, it is called once for each CMM installed - with the `CMMInfo` structure filled accordingly. The caller can pass a data reference to `CMIterateCMMInfo` which will then be passed to the `CMMIterateUPP`. This might be used to allow some of the information in the `CMMInfo` data structure to be put into a menu, for example, by passing a menu reference as the refcon. Either the proc or the count parameter must be provided. The caller will get a `paramErr` if both are nil.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMIterateColorDevices

Iterates through the color devices available on the system, returning device information to a callback you supply.

```
CMError CMIterateColorDevices (
   CMIterateDeviceInfoProcPtr proc,
   UInt32 *seed,
   UInt32 *count,
   void *refCon
);
```

Parameters

proc

A pointer to a function that iterates through device information available on the system. This is optional, but allows you to obtain device information. If provided, your callback is invoked once for each registered device.

seed

A pointer to a seed value. This is optional. If you pass a pointer to a seed value that is the same as the current seed value, then the callback function specified by the `proc` parameter is not invoked.

count

On output, the number of color devices available on the system.

refCon

An optional value that passed to your callback.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

This routine gets device information about all registered color devices. If provided, the supplied callback functions is called once for each registered device, passing in the device info and the supplied refcon.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMIterateColorSyncFolder

Iterates over the available profiles.

```
CMError CMIterateColorSyncFolder (
   CMProfileIterateUPP proc,
   UInt32 *seed,
   UInt32 *count,
   void *refCon
);
```

Parameters

proc

A universal procedure pointer of type `CMProfileIterateUPP`, which is described in `CMProfileIterateData` (page 923). If you do not wish to receive callbacks, pass `NULL` for this parameter. Otherwise, pass a pointer to your callback routine.

seed

A pointer to a value of type `long`. The first time you call `CMIterateColorSyncFolder`, you typically set the value to 0. In subsequent calls, you set the value to the seed value obtained from the previous call. ColorSync uses the value in determining whether to call your callback routine, as described in the discussion for this function.

On return, the value is the current seed for the profile cache (unless you pass `NULL`, as described in the discussion).

count

A pointer to a value of type `long`. On return, the value is the number of available profiles. `CMIterateColorSyncFolder` provides the number of profiles even when no iteration occurs (unless you pass `NULL`, as described in the discussion below). To determine the count alone, without iteration, call `CMIterateColorSyncFolder` and pass a value of `NULL` for all parameters except `count`.

refCon

An untyped pointer to arbitrary data supplied by your application. `CMIterateColorSyncFolder` passes this data to your callback routine. If you pass `NULL` for the `refCon` parameter, `CMIterateColorSyncFolder` passes `NULL` to your callback routine.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

Starting with ColorSync version 2.5, when your application needs information about the profiles currently available in the ColorSync Profiles folder, it can call the `CMIterateColorSyncFolder` routine, which in turn calls your callback routine once for each profile.

Even though there may be many profiles available, `CMIterateColorSyncFolder` can take advantage of ColorSync's profile cache to return profile information quickly, and (if the cache is valid) without having to open any profiles. For each profile, `CMIterateColorSyncFolder` supplies your routine with the profile header, script code, name, and location, in a structure of type `CMProfileIterateData` (page 923). As a result, your routine may be able to perform its function, such as building a list of profiles to display in a pop-up menu, without further effort (such as opening each file-based profile).

Only 2.x profiles are included in the profile search result.

Before calling `CMIterateColorSyncFolder` for the first time, you typically set `seed` to 0. ColorSync compares 0 to its current seed for the profile cache. It is not likely they will match—the odds are roughly one in two billion against it. If the values do not match, the routine iterates through all the profiles in the cache, calling your callback routine once for each profile. `CMIterateColorSyncFolder` then returns the actual seed value in `seed` (unless you passed `NULL` for that parameter).

If you pass the returned seed value in a subsequent call, and if there has been no change in the available profiles, the passed seed will match the stored cache seed and no iteration will take place.

Note that you can pass a `NULL` pointer for the `seed` parameter without harm. The result is the same as if you passed a pointer to 0, in that the function iterates through the available profiles, calling your callback routine once for each profile. However, the function does not return a seed value, since you have not passed a valid pointer.

You can force ColorSync to call your callback routine (if any profiles are available) by passing a `NULL` pointer or by passing 0 for the seed value. But suppose you have an operation, such as building a pop-up menu, that you only want to perform if the available profiles have changed. In that case, you pass the seed value from a previous call to `CMIterateColorSyncFolder`. If the profile folder has not changed, ColorSync will not call your callback routine.

Note that if there are no profiles available, ColorSync does not call your callback routine.

You can safely pass `NULL` for any or all of the parameters to the `CMIterateColorSyncFolder` function. If you pass `NULL` for all of the parameters, calling the function merely forces rebuilding of the profile cache, if necessary.

Version Notes
Starting with version 2.5, the name and location of the profile folder changed. In addition, the folder can now contain profiles within nested folders, as well as aliases to profiles or aliases to folders containing profiles. There are limits on the nesting of folders and aliases.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMIterateDeviceProfiles

Iterates through the device profiles available on the system and returns information about profiles of the devices to a callback you supply.

```
CMError CMIterateDeviceProfiles (
   CMIterateDeviceProfileProcPtr proc,
   UInt32 *seed,
   UInt32 *count,
   UInt32 flags,
   void *refCon
);
```

Parameters

proc

A pointer to a function that iterates through device information available on the system. This is optional, but allows you to obtain profile information for each device. If provided, your callback is invoked once for each registered device.

seed

A pointer to a seed value. This is optional. If you pass a pointer to a seed value that is the same as the current seed value, then the callback function specified by the `proc` parameter is not invoked.

count

On output, the number of color devices available on the system.

flags

A value that specifies which set of profiles you want to iterate through. It can have the following values: `cmIterateFactoryDeviceProfiles`, `cmIterateCustomDeviceProfiles`, `cmIterateCurrentDeviceProfiles`, `cmIterateAllDeviceProfiles` or `0`. Supplying `0` is the same as supplying `cmIterateCurrentDeviceProfiles`.

refCon

An optional value that passed to your callback.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMLaunchControlPanel

Launches the ColorSync preferences pane.

```
CMError CMLaunchControlPanel (
   UInt32 flags
);
```

Parameters

flags

A value that secifies how the preferences pane is launched. You currently must pass a vlaue of 0 for this parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
When your application calls the function `CMLaunchControlPanel`, any changes made by the user will not be available (through calls such as `CMGetDefaultProfileBySpace`) until the user closes the ColorSync preferences pane. There is currently no ColorSync function that determines if the ColorSync preferences pane has been closed, but you can use the Process Manager API for this purpose.

Availability
Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMLinkImage

Matches an image file with a device link profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMLinkImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 qual,
   CMProfileRef lnkProf,
   UInt32 lnkIntent
);
```

Parameters

specFrom

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

If a file with the same name already exists, it will be replaced if this parameter is set to true.

qual

The optional quality for the match—normal, draft or best (`cmNormalMode`, `cmDraftMode`, or `cmBestMode`).

lnkProf

The device link profile for the match.

lnkIntent

The rendering intent for the match—perceptual intent, relative colorimetric intent, saturation i ntent , or absolute colorimetric intent (`cmPerceptual`, `cmRelativecolorimetric`, `cmSaturation`, or `cmAbsoluteColorimetric`).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMMakeProfile

Makes a display or abstract profile by modifying an existing one.

```
CMError CMMakeProfile (
   CMProfileRef prof,
   CFDictionaryRef spec
);
```

Parameters

prof

The profile to modify.

spec

A dictionary that specifies the modifications to make to the profile supplied in the `prof` parameter.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The function `CMMakeProfile` adds appropriate tags to a profile to make a display or abstract profile based on a specification dictionary you supply.

One key in the specification dictionary must be `"profileType"` with a CFString value of either `"abstractLab"`, `"displayRGB"` or `"displayID"`.

The dictionary can optionally contain these keys-value pairs:

- `"description"`, with an associated CFString value
- `"copyright"`, with an associated CFString value

For a `profileType` key whose value is `"abstractLab"`, the dictionary can also contain the keys-value pairs listed in Table 32-1.

Table 32-1 Key-value pairs for "abstractLab"

Key	Value	Comment
`"gridPoints"`	A CFNumber (`SInt32`) that is an odd	Required
`"proc"`	A CFNumber (`SInt64`) coerced from a `LabToLabProcPtr` data type	Required
`"refcon"`	A CFNumber (`SInt64`) value coerced from a `void*` data type	Optional

For a `profileType` key whose value is `"displayRGB"`, the dictionary can also contain the keys-value pairs listed in Table 32-2.

Table 32-2 Key-value pairs for "displayRGB"

Key	Value	Comment
`"targetGamma"`	A CFNumber (`Float`), for example, 1.8	Optional
`"targetWhite"`	A CFNumber (`SInt32`), for example, 6500	Optional
`"gammaR`	A CFNumber (`Float`), for example, 2.5	Required
`"gammaG"`	A CFNumber (`Float`), for example, 2.5	Required

Key	Value	Comment
"gammaB"	A CFNumber (Float), for example, 2.5	Required
"tableChans"	A CFNumber (SInt32), for example, 1 or 3	Optional
"tableEntries"	A CFNumber (SInt32), for example, 16 or 255	Optional
"tableEntrySize"	A CFNumber (SInt32), for example,1 or 2	Optional
"tableData"	A CFData (lut in RRRGGGBBB order)	Optional
"phosphorRx"	A CFNumber (Float)	Only if not supplying the phospherSet key.
phosphorRy"	A CFNumber (Float)	Only if not supplying the phospherSet key.
phosphorGx"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorGy"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorBx"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorBy"	A CFNumber (Float)	Only if not supplying the phospherSet key.
"phosphorSet"	A CFString: "WideRGB", "700/525/450nm", "P22-EBU", "HDTV", "CCIR709", "sRGB", "AdobeRGB98" or "Trinitron"	Only if not supplying the phospher R, G, B keys
"whitePointx"	A CFNumber (Float)	Only if not supplying a whiteTemp key
"whitePointy"	A CFNumber (Float)	Only if not supplying a whiteTemp key
"whiteTemp"	A CFNumber (SInt32), for example, 5000, 6500, or 9300	Only if not supplying whitePointx and whitePointy keys

For a profileType key whose value is "displayID", the dictionary can also contain the keys-value pairs in Table 32-3

Table 32-3 Key-value pairs for “displayID”

Key	Value	Comment
"targetGamma"	A CFNumber (Float), for example, 1.8	Optional
"targetWhite"	A CFNumber (SInt32), for example, 6500	Optional

Key	Value	Comment
"displayID"	A CFNumber (SInt32)	Required

Optionally, the keys-value pairs s for a profileType key whose value is "displayRGB" can be provided to override the values from the display.

Availability
Available in Mac OS X v. 10.3 and later.

Declared In
CMApplication.h

CMMatchImage

Color matches an image file. (Deprecated in Mac OS X v10.5.)

```
CMError CMMatchImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 qual,
   CMProfileRef srcProf,
   UInt32 srcIntent,
   CMProfileRef dstProf
);
```

Parameters

specFrom
A file specification for the image file. See the File Manager documentation for a description of the FSSpec data type.

specInto
If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the FSSpec data type.

repl
A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

qual
The optional quality for the match—normal, draft or best (cmNormalMode, cmDraftMode, or cmBestMode).

srcProf
The optional source profile for the match.

srcIntent
The rendering intent for the match—perceptual intent, relative colorimetric intent, saturation i ntent , or absolute colorimetric intent (cmPerceptual, cmRelativecolorimetric, cmSaturation, or cmAbsoluteColorimetric).

dstProf
The destination profile for the match.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMNewProfile

Creates a new profile and associated backing copy.

```
CMError CMNewProfile (
   CMProfileRef *prof,
   const CMProfileLocation *theProfile
);
```

Parameters

prof

A pointer to a profile reference of type `CMProfileRef` (page 925). On return, a reference to the new profile.

theProfile

A pointer of type `CMProfileLocation` (page 924) to the profile location where the new profile should be created. A profile is commonly disk-file based—the disk file type for a profile is `'prof'`. However, to accommodate special requirements, you can create a handle- or pointer-based profile, you can create a temporary profile that is not saved after you call the `CMCloseProfile` function, or you can create a profile that is accessed through a procedure provided by your application. To create a temporary profile, you either specify `cmNoProfileBase` as the kind of profile in the profile location structure or specify `NULL` for this parameter.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMNewProfile` function creates a new profile and backing copy in the location you specify. After you create the profile, you must fill in the profile header fields and populate the profile with tags and their element data, and then call the function `CMUpdateProfile` (page 816) to save the element data to the profile file. The default ColorSync profile contents include a profile header of type `CM2Header` (page 875) and an element table.

To set profile elements outside the header, you use the function `CMSetProfileElement` (page 810), the function `CMSetProfileElementSize` (page 812), and the function `CMSetPartialProfileElement` (page 807). You set these elements individually, identifying them by their tag names.

When you create a new profile, all fields of the `CM2Header` profile header are set to 0 except the `size` and `profileVersion` fields. To set the header elements, you call the function `CMGetProfileHeader` (page 769) to get a copy of the header, assign values to the header fields, then call the function `CMSetProfileHeader` (page 813) to write the new header to the profile.

For each profile class, such as a device profile, there is a specific set of elements and associated tags, defined by the ICC, that a profile must contain to meet the baseline requirements. The ICC also defines optional tags that a particular CMM might use to optimize or improve its processing. You can also define private tags, whose tag signatures you register with the ICC, to provide a CMM with greater capability to refine its processing.

After you fill in the profile with tags and their element data, you must call the `CMUpdateProfile` function to write the new profile elements to the profile file.

This function is most commonly used by profile developers who create profiles for device manufacturers and by calibration applications. In most cases, application developers use existing profiles.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMNewProfileSearch

Searches the ColorSync Profiles folder and returns a list of 2.x profiles that match the search specification. (Deprecated in Mac OS X v10.5.)

```
CMError CMNewProfileSearch (
   CMSearchRecord *searchSpec,
   void *refCon,
   UInt32 *count,
   CMProfileSearchRef *searchResult
);
```

Parameters

searchSpec

A pointer to a search specification. For a description of the information you can provide in a search record of type `CMSearchRecord` to define the search, see `CMSearchRecord` (page 932). See the QuickDraw Reference for a description of the PixMap data type.

refCon

An untyped pointer to arbitrary data supplied by your application. `CMNewProfileSearch` passes this data to your filter routine. For a description of the filter routine, see the function `CMProfileFilterProcPtr` (page 864).

count

A pointer to a profile count. On return, a one-based count of profiles matching the search specification.

searchResult

A pointer to a search result reference. On return, a reference to the profile search result list. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 927). See the QuickDraw Reference for a description of the PixMap data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMNewProfileSearch` function sets up and defines a new search identifying through the search record the elements that a profile must contain to qualify for inclusion in the search result list. The function searches the ColorSync profiles folder for version 2.x profiles that meet the criteria and returns a list of these profiles in an internal private data structure whose reference is returned to you in the `searchResult` parameter.

You must provide a search record of type `CMSearchRecord` identifying the search criteria. You specify which fields of the search record to use for any given search through a search bit mask whose value you set in the search record's `searchMask` field.

Among the information you can provide in the search record is a pointer to a filter function to use to eliminate profiles from the search based on additional criteria not defined by the search record. The search result reference is passed to the filter function after the search is performed. For a description of the filter function and its prototype, see the function `CMProfileFilterProcPtr` (page 864).

Your application cannot directly access the search result list. Instead, you pass the returned search result list reference to other search-related functions that allow you to use the result list.

When your application has completed its search, it should call the function `CMDisposeProfileSearch` (page 745) to free the private memory allocated for the search.

To obtain a reference to a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfile` (page 798). To obtain the file specification for a profile corresponding to a specific index in the list, use the function `CMSearchGetIndProfileFileSpec` (page 799). To update the search result list, use the function `CMUpdateProfileSearch` (page 817). To free the private memory allocated for a profile search after your application has completed the search, use the function `CMDisposeProfileSearch` (page 745).

Version Notes
The `CMNewProfileSearch` function does not take full advantage of the optimized profile searching available starting with ColorSync version 2.5. Use `CMIterateColorSyncFolder` (page 780) instead.

This function is not recommended for use in ColorSync 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMOpenProfile

Opens the specified profile and returns a reference to the profile.

```
CMError CMOpenProfile (
   CMProfileRef *prof,
   const CMProfileLocation *theProfile
);
```

Parameters

prof

A pointer to a profile reference of type `CMProfileRef` (page 925). On return, the reference refers to the opened profile.

theProfile

A pointer to a profile location of type `CMProfileLocation` (page 924) for the profile to open. Commonly a profile is disk-file based, but it may instead be temporary, handle-based, pointer-based, or accessed through a procedure supplied by your application.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

If the `CMOpenProfile` function executes successfully, the profile reference refers to the opened profile. Your application uses this reference, for example, when it calls functions to color match, copy, and update a profile, and validate its contents.

The ColorSync Manager maintains private storage for each request to open a profile, allowing more than one application to use a profile concurrently.

When you create a new profile or modify the elements of an existing profile, the ColorSync Manager stores the new or modified elements in the private storage it maintains for your application. Any new or changed profile elements are not incorporated into the profile itself unless your application calls the function `CMUpdateProfile` (page 816) to update the profile. If you call the function `CMCopyProfile` (page 740) to create a copy of an existing profile under a new name, any changes you have made are incorporated in the profile duplicate but the original profile remains unchanged.

Before you call the `CMOpenProfile` function, you must set the `CMProfileLocation` data structure to identify the location of the profile to open. Most commonly, a profile is stored in a disk file. If the profile is in a disk file, use the profile location data type to provide its file specification. If the profile is in memory, use the profile location data type to specify a handle or pointer to the profile. If the profile is accessed through a procedure provided by your application, use the profile location data type to supply a universal procedure pointer to your procedure.

Your application must obtain a profile reference before you copy or validate a profile, and before you flatten the profile to embed it.

For example, your application can:

- open a profile
- call the `CMGetProfileHeader` function to obtain the profile's header to modify its values
- set new values
- call the `CMSetProfileHeader` function to replace the modified header
- pass the profile reference to a function such as `NCWNewColorWorld` (page 846) as the source or destination profile in a color world for a color-matching session
- When you close your reference to the profile by calling the function `CMCloseProfile` (page 728), your changes are discarded (unless you called the `CMUpdateProfile` function).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CMApplication.h`

CMProfileElementExists

Tests whether the specified profile contains a specific element based on the element's tag signature.

```
CMError CMProfileElementExists (
   CMProfileRef prof,
   OSType tag,
   Boolean *found
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) that specifies the profile to examine.

tag

The tag signature (for example, 'A2B0', or constant `cmAToB0Tag`) for the element in question. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

found

A pointer to a flag for whether the element was found. On return, the flag has the value `true` if the profile contains the element or `false` if it does not.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
You cannot use this function to test whether certain data in the `CM2Header` profile header exists. Instead, you must call the function `CMGetProfileHeader` (page 769) to copy the entire profile header and read its contents.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMProfileIdentifierFolderSearch

Searches the ColorSync Profiles folder and returns a list of profile references, one for each profile that matches the specified profile identifier. (Deprecated in Mac OS X v10.5.)

```
CMError CMProfileIdentifierFolderSearch (
   CMProfileIdentifierPtr ident,
   UInt32 *matchedCount,
   CMProfileSearchRef *searchResult
);
```

Parameters

ident

A pointer to a profile identifier structure specifying the profile to search for.

matchedCount

A pointer to a value of type `unsigned long`. On return, the one-based count of profiles that match the specified profile identifier. The count is typically 0 or 1, but can be higher.

searchResult

A pointer to a search result reference of type `CMProfileSearchRef` (page 927). On return, a reference to the profile search result list. Only version 2.x profiles are included in the profile search result.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020). It is not an error condition if this function finds no matching profiles. It returns an error only if a File Manager or other low-level system error occurs.

Discussion

When your application or device driver processes an image, it can keep a list of profile references for each profile it encounters in the image. Each time it encounters an embedded profile identifier, your application can call the function `CMProfileIdentifierListSearch` (page 793) to see if there is already a matching profile reference in its list. If not, it can call the `CMProfileIdentifierFolderSearch` function to see if the profile is located in the ColorSync Profiles folder.

Although there should typically be at most one profile in the ColorSync Profiles folder that matches the profile identifier, two or more profiles with different filenames may qualify.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMProfileIdentifierListSearch

Searches a list of profile references and returns a list of all references that match a specified profile identifier. (Deprecated in Mac OS X v10.5.)

```
CMError CMProfileIdentifierListSearch (
   CMProfileIdentifierPtr ident,
   CMProfileRef *profileList,
   UInt32 listSize,
   UInt32 *matchedCount,
   CMProfileRef *matchedList
);
```

Parameters

ident

A pointer to a profile identifier. The function looks for profile references in `profileList` that match the profile described by this identifier. For information on how a profile identifier match is determined, see `CMProfileIdentifier` (page 921).

profileList

A pointer to a list of profile references to search.

listSize

The number of profile references in `profileList`.

matchedCount

A pointer to a count of matching profile references. If you set `matchedList` to `NULL`, On return `matchedCount` specifies the number of references in `profileList` that match `ident`. The count is typically 0 or 1, but can be higher.

If you do not set `matchedList` to `NULL`, on input you set `matchedCount` to the maximum number of matching references to be returned in `matchedList`. On return, the value of `matchedCount` specifies the actual number of matching references returned, which is always equal to or less than the number passed in.

matchedList

A pointer to a list of profile references. If you set `matchedList` to `NULL` on input, On return nothing is returned in the parameter, and the actual number of matching references is returned in `matchedCount`.

If you do not set `matchedList` to `NULL` on input, it is treated as a pointer to allocated memory. On return, the allocated memory will contain a list, in no particular order, of profile references that match `ident`. Only version 2.x profiles are included in the profile search result. The number of references in the list is equal to or less than the value you pass in the `matchedCount` parameter. You must allocate enough memory for `matchedList` to store the requested number of profile references.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020). It is not an error condition if the `CMProfileIdentifierListSearch` function finds no matching profiles. The function returns an error only if a Memory Manager or other low-level system error occurs.

Discussion

When your application or device driver processes an image, it can keep a list of profile references for each unique profile or profile identifier it encounters in the image. Each time it encounters an embedded profile identifier, your application can call the `CMProfileIdentifierListSearch` function to see if there is already a matching profile reference in the list. Although your list of profile references would normally contain at most one reference that matches the profile identifier, it is possible to have two or more matches. For information on how a profile identifier match is determined, see `CMProfileIdentifier` (page 921).

If no matching profile is found in the list, your application can call the function `CMProfileIdentifierFolderSearch` (page 792) to see if a matching profile can be found in the ColorSync Profiles folder.

To determine the amount of memory needed for the list of profile references that match a profile identifier, your application may want to call `CMProfileIdentifierListSearch` twice. The first time, on input you set `matchedList` to `NULL` and ignore `matchedCount`. On return, `matchedCount` specifies the number of matching profiles. You then allocate enough memory to hold that many profile references (or a smaller number if you do not want all the references) and call `CMProfileIdentifierListSearch` again. This time you set `matchedList` to a pointer to the allocated memory and set `matchedCount` to the number of references you wish to obtain. To allocate memory, you use code such as the following:

```
myProfileRefListPtr = NewPtr(sizeof(CMProfileRef) * matchedCount);
```

If your application is interested in obtaining only the first profile that matches the specified profile, you need call `CMProfileIdentifierListSearch` only once. To do so, you just allocate enough memory to store one profile reference, set `matchedList` to point to that memory (or just set `matchedList` to point to a local variable), and set `matchedCount` to 1. On return, if `matchedCount` still has the value 1, then `CMProfileIdentifierListSearch` found a matching profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMProfileModified

Indicates whether the specified profile has been modified since it was created or last updated.

```
CMError CMProfileModified (
   CMProfileRef prof,
   Boolean *modified
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 925) to the profile to examine.

modified
A pointer to a Boolean variable. On return, the value of `modified` is set to `true` if the profile has been modified, `false` if it has not.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
When a profile is first opened, its modified flag is set to `false`. On calls that add to, delete from, or set the profile header or tags, the modified flag is set to `true`. After calling the function `CMUpdateProfile` (page 816), the modified flag is reset to `false`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMProofImage

Proofs an image. (Deprecated in Mac OS X v10.5.)

```
CMError CMProofImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 qual,
   CMProfileRef srcProf,
   UInt32 srcIntent,
   CMProfileRef dstProf,
   CMProfileRef prfProf
);
```

Parameters

specFrom
The destination profile for the match. See the File Manager documentation for a description of the `FSSpec` data type.

specInto
If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl
A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

qual
The optional quality for the match—normal, draft or best (`cmNormalMode`, `cmDraftMode`, or `cmBestMode`).

srcProf
The optional source profile for the match.

srcIntent
The rendering intent for the match—perceptual intent, relative colorimetric intent, saturation i ntent , or absolute colorimetric intent (`cmPerceptual`, `cmRelativecolorimetric`, `cmSaturation`, or `cmAbsoluteColorimetric`).

dstProf
The destination profile for the match.

prfProf
The proof profile for the match between the destination and proof profiles.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMRegisterColorDevice

Registers a device with ColorSync.

```
CMError CMRegisterColorDevice (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CFDictionaryRef deviceName,
   const CMDeviceScope *deviceScope
);
```

Parameters

deviceSpec
The class of the device (e.g., 'scnr' ,'cmra' ,'prtr' ,'mntr').

deviceScope
The unique identifier of the class (Class + ID uniquely id's device).

deviceName
Name of the device. See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

deviceScope
Structure defining the user and host scope this device pertains to.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
For a device to be recognized by ColorSync (and possibly other parts of Mac OS X) it needs to register itself using this function. If the device has ColorSync profiles associated with it, it should identify those u after registering with this function. Once a device is registered, it can appear as an input, output, or proofing device in ColorSync controls, as long as it has profiles associated with it. Registration need only happen once, when the device is installed. Device drivers need not register their device each time they are loaded.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMRemoveProfileElement

Removes an element corresponding to a specific tag from the specified profile.

```
CMError CMRemoveProfileElement (
   CMProfileRef prof,
   OSType tag
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile containing the tag remove.

tag

The tag signature for the element to remove.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMRemoveProfileElement` function deletes the tag as well as the element data from the profile.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSearchGetIndProfile

Opens the profile corresponding to a specific index into a specific search result list and obtains a reference to that profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMSearchGetIndProfile (
   CMProfileSearchRef search,
   UInt32 index,
   CMProfileRef *prof
);
```

Parameters

search

A reference to the profile search result list containing the profile whose reference you want to obtain. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 927). See the QuickDraw Reference for a description of the PixMap data type.

index

The position of the profile in the search result list. This value is specified as a one-based index into the set of profiles of the search result. The index must be less than or equal to the value returned as the `count` parameter of the `CMNewProfileSearch` function or the `CMUpdateProfileSearch` function; otherwise `CMSearchGetIndProfile` returns a result code of `cmIndexRangeErr`.

prof

A pointer to a profile reference of type `CMProfileRef` (page 925). On return, the reference refers to the profile associated with the specified index. See the QuickDraw Reference for a description of the PixMap data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Before your application can call the `CMSearchGetIndProfile` function, it must call the function `CMNewProfileSearch` (page 789) to perform a profile search and produce a search result list. The search result list is a private data structure maintained by the ColorSync Manager. After your application has finished using the profile reference, it must close the reference by calling the function `CMCloseProfile` (page 728).

Version Notes
This function is not recommended for use in ColorSync 2.5.

Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 780) for profile searching.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMSearchGetIndProfileFileSpec

Obtains the file specification for the profile at a specific index into a search result. (Deprecated in Mac OS X v10.5.)

```
CMError CMSearchGetIndProfileFileSpec (
   CMProfileSearchRef search,
   UInt32 index,
   FSSpec *spec
);
```

Parameters

search
A reference to the profile search result containing the profile whose file specification you want to obtain. For a description of the `CMProfileSearchRef` private data type, see `CMProfileSearchRef` (page 927). See the QuickDraw Reference for a description of the PixMap data type.

index
The index of the profile whose file specification you want to obtain. This is a one-based index into a set of profiles in the search result list. The index must be less than or equal to the value returned as the `count` parameter of the `CMNewProfileSearch` function or the `CMUpdateProfileSearch` function; otherwise `CMSearchGetIndProfile` returns a result code of `cmIndexRangeErr`.

profileFile
A pointer to a file specification. On return, this parameter points to a file specification for the profile at the location specified by `index`. See the QuickDraw Reference for a description of the PixMap data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Before your application can call the `CMSearchGetIndProfileFileSpec` function, it must call the function `CMNewProfileSearch` (page 789) to perform a profile search and produce a search result list. The search result list is a private data structure maintained by ColorSync.

The `CMSearchGetIndProfileFileSpec` function obtains the Macintosh file system file specification for a profile at a specific index in the search result list.

Version Notes
This function is not recommended for use in ColorSync 2.5.

Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 780) for profile searching.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMSetDefaultDevice

Sets the default device.

```
CMError CMSetDefaultDevice (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID
);
```

Parameters
deviceClass
The class of the device (e.g., 'scnr' ,'cmra' ,'prtr' ,'mntr').
deviceID
The unique identifier of the class (Class + ID uniquely id's device).

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
For each class of device, a device management layer may establish which of the registered devices is the default. This helps keep color management choices to a minimum and allows for some "automatic" features to be enabled, such as, "Default printer" as an output profile selection. If no such device (as specified by deviceClass and deviceID) has been registered, an error is returned.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMSetDefaultProfileBySpace

Sets the default profile for the specified color space. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetDefaultProfileBySpace (
   OSType dataColorSpace,
   CMProfileRef prof
);
```

Parameters

dataColorSpace

A four-character identifier of type `OSType`. You pass a color space signature that identifies the color space you wish to set the default profile for. The currently-supported values are `cmRGBData`, `cmCMYKData`, `cmLabData`, and `cmXYZData`. These constants are a subset of the constants described in "Color Space Signatures" (page 969). If you supply a value that is not supported, the `CMGetDefaultProfileBySpace` function returns an error value of `paramErr`.

prof

A profile reference. Before calling `CMSetDefaultProfileBySpace`, set the reference to specify the default profile for the color space. The profile must be file-based; otherwise, the function returns a `CMInvalidProfileLocation` error.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CMSetDefaultProfileBySpace` function currently supports the RGB, CMYK, Lab, and XYZ color spaces. The signature constants for these color spaces (shown above with the `dataColorSpace` parameter description) are described in "Color Space Signatures" (page 969). Support for additional color spaces may be provided in the future. `CMSetDefaultProfileBySpace` returns a value of `paramErr` if you pass a color space constant it does not currently support.

Note that a user can also use the ColorSync control panel to specify a default profile for the RGB and CMYK color spaces.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMSetDefaultProfileByUse

Sets values for device profile settings. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetDefaultProfileByUse (
   OSType use,
   CMProfileRef prof
);
```

Parameters

use

A value that specifies the device type for which to set the profile.

prof

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMSetDeviceDefaultProfileID

Sets the default profile ID for a given device.

```
CMError CMSetDeviceDefaultProfileID (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID defaultProfID
);
```

Parameters

deviceClass

The device class for the device whose default profile you want to set. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

The device ID for the device whose default profile you want to set.

defaultID

The ID of profile you want to set as the default.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The default profile ID for a given device is an important piece of information because of the function `CMGetProfileByUse`. The function `CMGetProfileByUse` returns the default profile for devices depending on the user's selection in the ColorSync preferences pane. Device drivers and host software can set the default profile for a given device using the function `CMSetDeviceDefaultProfileID`.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In
CMDeviceIntegration.h

CMSetDeviceFactoryProfiles

Establishes the profiles used by a given device.

```
CMError CMSetDeviceFactoryProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceProfileID defaultProfID,
   const CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass
The device class for the device whose factory profiles you want to establish. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID
The device ID for the device whose factory profiles you want to establish.

defaultProfID
The ID of the default profile for this device.

deviceProfiles
On output, points to array that contains the factory device profiles.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
This function establishes the profiles used by a given device. It should be called after device registration to notify ColorSync of the device's profiles. Note that factory device profiles and the current device profiles might not be the same, since the latter may contain modifications to the factory set.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
CMDeviceIntegration.h

CMSetDeviceProfile

Change the profile used by a given device.

```
CMError CMSetDeviceProfile (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   const CMDeviceProfileScope *profileScope,
   CMDeviceProfileID profileID,
   const CMProfileLocation *profileLoc
);
```

Parameters

deviceClass

The device class for the device whose profile you want to set. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

The device ID for the device whose profile you want to set.

profileScope

A pointer to the structure defining the scope this profile pertains to.

profileID

The ID of the default profile for this device.

deviceProfLoc

A pointer to the `CMProfileLocation` of the profile. Since this structure is a fixed length structure, you can simply pass a pointer to a stack-based structure or memory allocated for it.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

This function provides a way to change a profile used by a given device by ID. It can be called after device registration by calibration applications to reset a device's profile from factory defaults to calibrated profiles. In order for this call to be made successfully, you must pass the `CMDeviceClass` and `CMDeviceID` of the device being calibrated along with the `CMDeviceProfileID` of the profile to set. (Device selection and identification can be facilitated using the function `CMIterateColorDevices`). If an invalid `CMDeviceClass` or `CMDeviceID` is passed, an error (`CMInvalidDeviceClass` or `CMInvalidDeviceID`) is returned.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMSetDeviceProfiles

Changes the profiles used by a given device. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetDeviceProfiles (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   const CMDeviceProfileScope *profileScope,
   const CMDeviceProfileArray *deviceProfiles
);
```

Parameters

deviceClass

The device class for the device whose profiles you want to set. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

The device ID for the device whose profiles you want to set.

profileScope

A pointer to the structure defining the scope these profiles pertain to.

deviceProfiles

A pointer to the profile array that contains replacements for the factory profiles. You don't have to replace all the original profiles with this call. The array can contain as few as one profile or as many profiles as there are in the original factory array. You supply only those profiles you want to replace. Profiles are replaced by ID.

Return Value

A `CMError` value. If you pass a n invalid `CMDeviceClass` or `CMDeviceID`, the function returns `CMInvalidDeviceClass` or `CMInvalidDeviceID`. See "ColorSync Manager Result Codes" (page 1020).

Discussion

This function provides a way to change the profiles used by a given device. It can be called after device registration by calibration applications to reset a device's profiles from factory defaults to calibrated profiles. In order for this call to be made successfully, the caller must pass the `CMDeviceClass` and `CMDeviceID` device being calibrated. (You can call the function `CMIterateColorDevices` to find available device classes and IDs.).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMDeviceIntegration.h`

CMSetDeviceState

Sets the state of a device.

```
CMError CMSetDeviceState (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID,
   CMDeviceState deviceState
);
```

Parameters

deviceClass

The device class for the device whose state you want to set. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

The device ID for the device whose state you want to set.

deviceState

The device state to set. See "Device States" (page 980) for the values you can supply.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

This routines provides access for the device management layer to update the state of a particular device. For example, a device can be offline, busy, or calibrated. The state data passed in replaces the old state data with the value you supply.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.1 and later.

Declared In

`CMDeviceIntegration.h`

CMSetGammaByAVID

Sets the gamma for the specified display device.

```
CMError CMSetGammaByAVID (
   CMDisplayIDType theID,
   CMVideoCardGamma *gamma
);
```

Parameters

theID

A Display Manager ID value. You pass the ID value for the display device for which to set the gamma.

gamma

A pointer to the gamma value to which you want to set the display device.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 3.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetIndImageProfile

Sets a specific embeded profile for a given image. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetIndImageProfile (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl,
   UInt32 index,
   CMProfileRef prof
);
```

Parameters

specFrom

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If this parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

index

The numeric index of the profile to set.

prof

The profile to set at the index designated by the `index` parameter.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`CMScriptingPlugin.h`

CMSetPartialProfileElement

Sets part of the element data for a specific tag in the specified profile.

```
CMError CMSetPartialProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 offset,
   UInt32 byteCount,
   const void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile containing the tag for which the element data is set.

tag

The tag signature for the element whose data is set. The tag identifies the element. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

offset

The offset in the existing element data where data transfer should begin.

byteCount

The number of bytes of element data to transfer.

elementData

A pointer to the buffer containing the element data to transfer to the profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

You can use the `CMSetPartialProfileElement` function to set the data for an element when the amount of data is large and you need to copy it to the profile in segments.

After you set the element size, you can call this function repeatedly, as many times as necessary, each time appending a segment of data to the end of the data already copied until all the element data is copied.

If you know the size of the element data, you should call the function `CMSetProfileElementSize` (page 812) to reserve it before you call `CMSetPartialProfileElement` to set element data in segments. Setting the size first avoids the extensive overhead required to increase the size for the element data with each call to append another segment of data.

To copy the entire data for an element as a single operation, when the amount of data is small enough to allow this, call the function `CMSetProfileElement` (page 810).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetProfileByAVID

Sets the profile for the specified monitor, optionally setting video card gamma.

```
CMError CMSetProfileByAVID (
   CMDisplayIDType theID,
   CMProfileRef prof
);
```

Parameters

theAVID

A Display Manager ID value. You pass the ID value for the monitor for which to set the profile.

prof

A profile reference. Before calling `CMSetProfileByAVID`, set the reference to identify the profile for the monitor specified by `theAVID`.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

If you specify a profile that contains the optional profile tag for video card gamma, `CMSetProfileByAVID` extracts the tag and sets the video card based on the tag data. This is the only ColorSync function that sets video card gamma. The tag constant `cmVideoCardGammaTag` is described in "Video Card Gamma Tags" (page 1019).

When a user sets a display profile using the Monitors & Sound control panel, the system profile is set to the same profile. When you call `CMSetProfileByAVID` to set a profile for a monitor, you may also wish to make that profile the system profile. If so, you must call `CMSetSystemProfile` (page 814) explicitly—calling `CMSetProfileByAVID` alone has no affect on the system profile.

Note that if the Display Manager supports ColorSync, the `CMSetProfileByAVID` function calls on the Display Manager to set the profile for the specified display. This is the case if the version of the Display Manager is 2.2.5 or higher (if `gestaltDisplayMgrAttr` has the `gestaltDisplayMgrColorSyncAware` bit set).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetProfileDescriptions

Sets the description tag data for a specified profile.

```
CMError CMSetProfileDescriptions (
   CMProfileRef prof,
   const char *aName,
   UInt32 aCount,
   ConstStr255Param mName,
   ScriptCode mCode,
   const UniChar *uName,
   UniCharCount uCount
);
```

Parameters

prof

A reference to the profile into which to set the description tag data.

aName

A pointer to a 7-bit Roman ASCII profile name string to be set for the profile. This string must be null-terminated.

aCount

A count of the number of characters in the string specified in the `aName` parameter

mName

A pointer to the localized profile name string in Mac script-code format which is to be set for the profile. This string must be null-terminated.

mCode

The script code corresponding to the string specified by the `mName` parameter.

uName

A pointer to the localized Unicode profile name string which is to be set for the profile. This string must be null-terminated

uCount

A count of the number of Unicode characters in string specified by the `uName` parameter. Do not confuse this with a byte count, because each Unicode character requires two bytes.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

Use this function to set the description tag data for a given profile. The ICC Profile Format Specification (available at http://www.color.org) includes a description tag (`'desc'`), designed to provide more information about a profile than can be contained in a file name. This is especially critical on file systems with 8.3 names. The tag data can consist of up to three separate pieces (strings) of information for a profile. These different strings are designed to allow for display in different languages or on different computer systems. Applications typically use one of the strings to show profiles in a list or a pop-up menu.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetProfileElement

Sets or replaces the element data for a specific tag in the specified profile.

```
CMError CMSetProfileElement (
   CMProfileRef prof,
   OSType tag,
   UInt32 elementSize,
   const void *elementData
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile containing the tag for which the element data is set.

tag

The tag signature for the element whose data is set. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

elementSize

The size in bytes of the element data set.

elementData

A pointer to the buffer containing the element data to transfer to the profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMSetProfileElement` function replaces existing element data if an element with the specified tag is already present in the profile. Otherwise, it sets the element data for a new tag. Your application is responsible for allocating memory for the buffer to hold the data to transfer.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetProfileElementReference

Adds a tag to the specified profile to refer to data corresponding to a previously set element.

```
CMError CMSetProfileElementReference (
   CMProfileRef prof,
   OSType elementTag,
   OSType referenceTag
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile to add the tag to.

elementTag

The original element's signature tag corresponding to the element data to which the new tag will refer.

referenceTag

The new tag signature to add to the profile to refer to the element data corresponding to `elementTag`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
After the `CMSetProfileElementReference` function executes successfully, the specified profile will contain more than one tag corresponding to a single piece of data. All of these tags are of equal importance. Your application can set a reference to an element that was originally a reference itself without circularity.

If you call the function `CMSetProfileElement` (page 810) subsequently for one of the tags acting as a reference to another tag's data, then the element data you provide is set for the tag and the tag is no longer considered a reference. Instead, the tag corresponds to its own element data and not that of another tag.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetProfileElementSize

Reserves the element data size for a specific tag in the specified profile before setting the element data.

```
CMError CMSetProfileElementSize (
   CMProfileRef prof,
   OSType tag,
   UInt32 elementSize
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 925) to the profile in which the element data size is reserved.

tag
The tag signature for the element whose size is reserved. The tag identifies the element. For a complete list of the tag signatures a profile may contain, including a description of each tag, refer to the International Color Consortium Profile Format Specification. The signatures for profile tags are defined in the `CMICCProfile.h` header file.

elementSize
The total size in bytes to reserve for the element data.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Your application can use the `CMSetProfileElementSize` function to reserve the size of element data for a specific tag before you call the function `CMGetPartialProfileElement` (page 765) to set the element data. The most efficient way to set a large amount of element data when you know the size of the data is to first set the size, then call the `CMSetPartialProfileElement` function to set each of the data segments. Calling the `CMSetProfileElementSize` function first eliminates the need for the ColorSync Manager to repeatedly increase the size for the data each time you call the `CMSetPartialProfileElement` function.

In addition to reserving the element data size, the `CMSetProfileElementSize` function sets the element tag, if it does not already exist.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMSetProfileHeader

Sets the header for the specified profile.

```
CMError CMSetProfileHeader (
   CMProfileRef prof,
   const CMAppleProfileHeader *header
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile whose header is set.

header

A pointer to the new header to set for the profile.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

You can use the `CMSetProfileHeader` function to set a header for a version 1.0 or a version 2.x profile. Before you call this function, you must set the values for the header, depending on the version of the profile. For a version 2.x profile, you use a data structure of type `CM2Header` (page 875). For a version 1.0 profile, you use a data structure of type `CMHeader` (page 898). You pass the header you supply in the `CMAppleProfileHeader` union, described in `CMAppleProfileHeader` (page 881).

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`CMApplication.h`

CMSetProfileLocalizedStringDictionary

Writes a dictionary of localized strings to a given tag in a profile.

```
CMError CMSetProfileLocalizedStringDictionary (
   CMProfileRef prof,
   OSType tag,
   CFDictionaryRef theDict
);
```

Parameters

prof

The profile to modify.

tag

The tag type of profile to modify.

theDict

The dictionary to modify. See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMApplication.h`

CMSetSystemProfile

Sets the current system profile. (Deprecated in Mac OS X v10.5.)

```
CMError CMSetSystemProfile (
   const FSSpec *profileFileSpec
);
```

Parameters

profileFileSpec

A pointer to a file specification structure. Before calling `CMSetSystemProfile`, set the structure to specify the desired system profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
By default, a standard RGB profile is configured as the system profile. By calling the `CMSetSystemProfile` function, your application can specify a new system profile. You can configure only a display device profile as the system profile.

Version Notes
Starting with version 2.5, use of the system profile has changed.

The function `CMSetSystemProfile` does not retrieve video card gamma data (introduced in ColorSync version 2.5) to set the video card; use the function `CMSetProfileByAVID` (page 808) instead.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMUnembedImage

Removes any ICC profiles embeded in an image. (Deprecated in Mac OS X v10.5.)

```
CMError CMUnembedImage (
   const FSSpec *specFrom,
   const FSSpec *specInto,
   Boolean repl
);
```

Parameters

specFrom

A file specification for the image file. See the File Manager documentation for a description of the `FSSpec` data type.

specInto

If this parameter is a file, it specifies the resulting image. If this parameter is a folder, it specifies the location of the resulting image which will have the same name as the original file. If his parameter is not provided, the original file is modified. See the File Manager documentation for a description of the `FSSpec` data type.

repl

A Boolean value. If a file with the same name already exists, it will be replaced if this parameter is set to true.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CMUnregisterColorDevice

Unregisters a device.

```
CMError CMUnregisterColorDevice (
   CMDeviceClass deviceClass,
   CMDeviceID deviceID
);
```

Parameters

deviceClass

The device class of the device you want to unregister. See "Device Classes" (page 979) for a list of the constants you can supply.

deviceID

The device ID of the device you want to unregister.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
When a device is no longer to be used on a system (as opposed to being offline), it should be unregistered. If a device is temporarily shut down or disconnected, it does not to be unregistered unless either of the following is true:

- The device driver is being removed (uninstalled)
- The device driver can't access the device profiles without the device

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMUpdateProfile

Saves modifications to the specified profile.

```
CMError CMUpdateProfile (
   CMProfileRef prof
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile to update.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CMUpdateProfile` function makes permanent any changes or additions your application has made to the profile identified by the profile reference, if no other references to that profile exist.

The ColorSync Manager maintains a modified flag to track whether a profile has been modified. After updating a profile, the `CMUpdateProfile` function sets the value of the modified flag for that profile to `false`.

Each time an application calls the function `CMOpenProfile` (page 790), the function creates a unique reference to the profile. An application can also duplicate a profile reference by passing a copy to another task. You cannot use the `CMUpdateProfile` function to update a profile if more than one reference to the profile exists—attempting to do so will result in an error return. You can call the function `CMGetProfileRefCount` (page 772) to determine the reference count for a profile reference.

You cannot use the `CMUpdateProfile` function to update a ColorSync 1.0 profile.

After you fill in tags and their data elements for a new profile created by calling the function `CMNewProfile` (page 788), you must call the `CMUpdateProfile` function to write the element data to the new profile.

If you modify an open profile, you must call `CMUpdateProfile` to save the changes to the profile file before you call the function `CMCloseProfile` (page 728). Otherwise, the changes are discarded.

To modify a profile header, you use the function `CMGetProfileHeader` (page 769) and the function `CMSetProfileHeader` (page 813).

To set profile elements outside the header, you use the function `CMSetProfileElement` (page 810), the function `CMSetProfileElementSize` (page 812), and the function `CMSetPartialProfileElement` (page 807).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CMUpdateProfileSearch

Searches the ColorSync Profiles folder and updates an existing search result obtained originally from the CMNewProfileSearch function. (Deprecated in Mac OS X v10.5.)

```
CMError CMUpdateProfileSearch (
   CMProfileSearchRef search,
   void *refCon,
   UInt32 *count
);
```

Parameters

search

A reference to a search result list returned to your application when you called the CMNewProfileSearch function. For a description of the CMProfileSearchRef private data type, see CMProfileSearchRef (page 927). See the QuickDraw Reference for a description of the PixMap data type.

refCon

A pointer to a reference constant for application data passed as a parameter to calls to the filter function specified by the original search specification. For a description of the filter function, see the function CMProfileFilterProcPtr (page 864).

count

A pointer to a profile count. On return, if the function result is noErr, a one-based count of the number of profiles matching the original search specification passed to the CMNewProfileSearch function. Otherwise undefined.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
After a profile search has been set up and performed through a call to the CMNewProfileSearch function, the CMUpdateProfileSearch function updates the existing search result. You must use this function if the contents of the ColorSync Profiles folder have changed since the original search result was created.

The search update uses the original search specification, including the filter function indicated by the search record. Data given in the CMUpdateProfileSearch function's refCon parameter is passed to the filter function each time it is called.

Sharing a disk over a network makes it possible for modification of the contents of the ColorSync Profiles folder to occur at any time.

For a description of the function you call to begin a new search, see the function CMNewProfileSearch (page 789). That function specifies the filter function referred to in the description of the refCon parameter.

Version Notes
Starting with version 2.5, you should use the function `CMIterateColorSyncFolder` (page 780) for profile searching.

This function is not recommended for use in ColorSync 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMValidateProfile

Indicates whether the specified profile contains the minimum set of elements required by the current color management module (CMM) for color matching or color checking.

```
CMError CMValidateProfile (
   CMProfileRef prof,
   Boolean *valid,
   Boolean *preferredCMMnotfound
);
```

Parameters

prof
A profile reference of type `CMProfileRef` (page 925) to the profile to validate.

valid
A pointer to a valid profile flag. On return, has the value `true` if the profile contains the minimum set of elements to be valid and `false` if it does not.

preferredCMMnotfound
A pointer to a flag for whether the preferred CMM was found. On return, has the value `true` if the CMM specified by the profile was not available to perform validation or does not support this function and the default CMM was used. Has the value `false` if the profile's preferred CMM is able to perform validation.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
When your application calls the `CMValidateProfile` function, the ColorSync Manager dispatches the function to the CMM specified by the `CMMType` header field of the profile whose reference you specify. The preferred CMM can support this function or not.

If the preferred CMM supports this function, it determines if the profile contains the baseline elements for the profile class, which the CMM requires to perform color matching or gamut checking. For each profile class, such as a device profile, there is a specific set of required tagged elements defined by the ICC that the profile must include. The ICC also defines optional tags, which may be included in a profile. A CMM might use these optional elements to optimize or improve its processing. Additionally, a profile might include private tags defined to provide a CMM with processing capability particular to the needs of that CMM. The

profile developer can define these private tags, register the tag signatures with the ICC, and include the tags in a profile. The CMM checks only for the existence of profile elements it does not check the element's content and size.

If the preferred CMM does not support the `CMValidateProfile` function request, the ColorSync Manager calls the default CMM to handle the validation request.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CMValidImage

Validates the specified image file. (Deprecated in Mac OS X v10.5.)

```
CMError CMValidImage (
   const FSSpec *spec
);
```

Parameters

spec
A file specification for the image file you want to validate. See the File Manager documentation for a description of the `FSSpec` data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
This function validates the specified image file. ColorSync checks with any installed scripting plug-ins to see if they recognize the image's file format. If a scripting plug-in is found which recognizes the image's file format, `CMValidateImage` returns `noErr` . If the image's file format is not recognized, `CMValidateImage` returns the `cmInvalidImageFile` error.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMScriptingPlugin.h`

CWCheckBitmap

Tests the colors of the pixel data of a bitmap to determine whether the colors map to the gamut of the destination device.

```
CMError CWCheckBitmap (
   CMWorldRef cw,
   const CMBitmap *bitmap,
   CMBitmapCallBackUPP progressProc,
   void *refCon,
   CMBitmap *resultBitmap
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 942) to use for the color check.

The functions `NCWNewColorWorld` (page 846) and `CWConcatColorWorld` (page 823) both allocate color world references of type `CMWorldRef` (page 942).

bitmap

A pointer to a bitmap of type `CMBitmap` (page 882) whose colors are to be checked.

progressProc

A calling program–supplied callback function that allows your application to monitor progress or abort the operation as the bitmap's colors are checked against the gamut of the destination device. The default CMM calls your function approximately every half-second unless color checking occurs in less time this happens when there is a small amount of data to be checked. If the function returns a result of `true`, the operation is aborted. Specify `NULL` for this parameter if your application will not monitor the bitmap color checking. For information on the callback function and its type definition, see the function `CMBitmapCallBackProcPtr` (page 852).

refCon

A pointer to a reference constant for application data passed as a parameter to calls to `progressProc`.

resultBitmap

A pointer to a bitmap. On return, contains the results of the color check. The bitmap must have bounds equal to the parameter of the source bitmap pointed to by `bitMap`. You must allocate the pixel buffer pointed to by the `image` field of the structure `CMBitmap` (page 882) and initialize the buffer to zeroes. Pixels are set to 1 if the corresponding pixel of the source bitmap indicated by `bitMap` is out of gamut. You must set the `space` field of the `CMBitMap` structure to `cmGamutResult1Space` color space storage format, as described in "Abstract Color Space Constants" (page 946).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

When your application calls the `CWCheckBitMap` function, the ColorSync Manager dispatches the function to the preferred CMM. The ColorSync Manager determines the preferred CMM based on the color world configuration. If the color world you pass in was created by the `CWConcatColorWorld` function, then the `keyIndex` field of the `CMConcatProfileSet` data structure identifies the preferred CMM. If the preferred CMM is not available, the default CMM is used to perform the color matching.

For the `CWCheckBitMap` function to execute successfully, the source profile's `dataColorSpace` field value and the `space` field value of the source bitmap pointed to by the `bitMap` parameter must specify the same data color space. `CWCheckBitMap` is not supported if the color world was initialized with a named color space profile.

Availability

Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CWCheckColors

Tests a list of colors using a specified color world to see if they fall within the gamut of a destination device.

```
CMError CWCheckColors (
   CMWorldRef cw,
   CMColor *myColors,
   size_t count,
   UInt8 *result
);
```

Parameters

cw

A reference to the color world of type CMWorldRef (page 942) describing how the test is to occur.

The functions NCWNewColorWorld (page 846) and CWConcatColorWorld (page 823) both allocate color world references of type CMWorldRef (page 942).

myColors

A pointer to an array containing a list of colors of type CMColor (page 884) to be checked . This function assumes the color values are specified in the data color space of the source profile.

count

The number of colors in the array. This is a one-based count.

result

A pointer to a buffer of packed bits. On return, each bit value is interpreted as a bit field with each bit representing a color in the array pointed to by myColors. You allocate enough memory to allow for 1 bit to represent each color in the myColors array. Bits in the result field are set to 1 if the corresponding color is out of gamut for the destination device. Ensure that the buffer you allocate is zeroed out before you call this function.

To access the packed bit-array, use code similar to the following:

```
inline bool GetNthBit (UInt8* result, int n)
{
    return ( 0 != (result[n/8] & (128>>(n%8))) );
}
```

The result bit array indicates whether the colors in the list are in or out of gamut for the destination profile. If a bit is set, its corresponding color falls out of gamut for the destination device. The leftmost bit in the field corresponds to the first color in the list.

Return Value
A CMError value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The color test provides a preview of color matching using the specified color world.

All CMMs must support the CWCheckColors function.

If you have set a profile's gamut-checking mask so that no gamut information is included—see "Flag Mask Definitions for Version 2.x Profiles" (page 983) — CWCheckColors returns the cmCantGamutCheckError error.

The CWCheckColors function supports matching sessions set up with one of the multichannel color data types. CWCheckColors is not supported if the color world was initialized with a named color space profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
CMApplication.h

CWCheckPixMap

Checks the colors of a pixel map using the profiles of a specified color world to determine whether the colors are in the gamut of the destination device. (Deprecated in Mac OS X v10.4.)

```
CMError CWCheckPixMap (
   CMWorldRef cw,
   PixMap *myPixMap,
   CMBitmapCallBackUPP progressProc,
   void *refCon,
   BitMap *resultBitMap
);
```

Parameters

cw

A reference to the color world of type CMWorldRef (page 942) in which color checking is to occur.

The functions NCWNewColorWorld (page 846) and CWConcatColorWorld (page 823) both return color world references of type CMWorldRef (page 942).

See the QuickDraw Reference for a description of the PixMap data type.

myPixMap

A pointer to the pixel map to check colors for. A pixel map is a QuickDraw structure describing pixel data. The pixel map must be nonrelocatable; to ensure this, you should lock the handle to the pixel map. See the QuickDraw Reference for a description of the PixMap data type.

progressProc

A calling program–supplied callback function that allows your application to monitor progress or abort the operation as the pixel map colors are checked against the gamut of the destination device.

The default CMM calls your function approximately every half-second unless color checking occurs in less time; this happens when there is a small amount of data to be checked. If the function returns a result of true, the operation is aborted. Specify NULL for this parameter if your application will not monitor the pixel map color checking. For information on the callback function and its type definition, see the function CMBitmapCallBackProcPtr (page 852).

See the QuickDraw Reference for a description of the PixMap data type.

refCon

A pointer to a reference constant for application data passed as a parameter to calls to your CMBitmapCallBack function pointed to by progressProc.

resultBitMap

A pointer to a QuickDraw bitmap. On return, bits are set to 1 if the corresponding pixel of the pixel map indicated by myPixMap is out of gamut. Boundaries of the bitmap indicated by resultBitMap must equal the parameter of the pixel map indicated by the myPixMap. See the QuickDraw Reference for a description of the PixMap data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CWCheckPixMap` function performs a gamut test of the pixel data of the `myPixMap` pixel map to determine if its colors are within the gamut of the destination device as specified by the destination profile. The gamut test provides a preview of color matching using the specified color world.

The preferred CMM, as determined by the ColorSync Manager based on the profiles of the color world configuration, is called to perform the color matching.

If the preferred CMM is not available, then the ColorSync Manager calls the default CMM to perform the matching. If the preferred CMM is available but does not implement the `CMCheckPixmap` function, then the ColorSync Manager unpacks the colors in the pixel map to create a color list and calls the preferred CMM's `CMCheckColors` function, passing to this function the list of colors to match. Every CMM must support the `CMCheckColors` function.

For this function to execute successfully, the source and destination profiles' data color spaces (`dataColorSpace` field) must be RGB to match the data color space of the pixel map, which is implicitly RGB.

If you specify a pointer to a callback function in the `progressProc` parameter, the CMM performing the color checking calls your function to monitor progress of the session. Each time the CMM calls your function, it passes the function any data you specified in the `CWCheckPixMap` function's `refCon` parameter.

You can use the reference constant to pass in any kind of data your callback function requires. For example, if your application uses a dialog box with a progress bar to inform the user of the color-checking session's progress, you can use the reference constant to pass the dialog box's window reference to the callback routine. For information about the callback function, see the function `CMBitmapCallBackProcPtr` (page 852).

You should ensure that the buffer pointed to by the `baseAddr` field of the bitmap passed in the `resultBitMap` parameter is zeroed out.

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CWConcatColorWorld

Sets up a color world that includes a set of profiles for various color transformations among devices in a sequence.

```
CMError CWConcatColorWorld (
   CMWorldRef *cw,
   CMConcatProfileSet *profileSet
);
```

Parameters

cw

A pointer to a color world. On return, a reference to a color world of type `CMWorldRef` (page 942). You pass the returned reference to other functions that use the color world for color-matching and color-checking sessions.

profileSet

A pointer of type `CMConcatProfileSet` (page 887) to an array of profiles describing the processing to carry out. You create the array and initialize it in processing order—source through destination.

You set the `keyIndex` field of the `CMConcatProfileSet` data structure to specify the zero-based index of the profile within the profile array whose specified CMM should be used for the entire color-matching or color-checking session. The profile header's `CMMType` field specifies the CMM. This CMM will fetch the profile elements necessary for the session.

Note that starting with ColorSync 2.5, the user can set a preferred CMM with the ColorSync control panel. If that CMM is available, ColorSync will use that CMM for all color conversion and matching operations the CMM is capable of performing.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CWConcatColorWorld` function sets up a session for color processing that includes a set of profiles. The array of profiles is in processing order—source through destination. Your application passes the function a pointer to a data structure of type `CMConcatProfileSet` to identify the profile array.

The quality flag setting—indicating normal mode, draft mode, or best mode—specified by the first profile prevails for the entire session the quality flags of following profiles in the sequence are ignored. The quality flag setting is stored in the `flags` field of the profile header. See `CM2Header` (page 875) and "Flag Mask Definitions for Version 2.x Profiles" (page 983) for more information on the use of flags.

The rendering intent specified by the first profile is used to color match to the second profile, the rendering intent for the second profile is used to color match to the third profile, and so on through the series of concatenated profiles.

The following rules govern the profiles you can specify in the profile array pointed to by the `profileSet` parameter for use with the `CWConcatColorWorld` function:

- In the profile array, you can pass in one or more profiles, but you must specify at least one profile. If you specify only one profile, it must be a device link profile. If you specify a device link profile, you cannot specify any other profiles in the profiles array; a device link profile must be used alone.
- In the profile array, you can specify an abstract profile anywhere in the sequence other than as the first or last profile.
- For the first and last profiles, you can specify device profiles or color space conversion profiles. However, when you set up a color-matching session with a named color space profile and other profiles, the named color profile must be first or the last profile in the color world—it cannot be in the middle.
- You cannot specify `NULL` to indicate the system profile. Note that starting with version 2.5, use of the system profile has changed.
- If you specify a color space profile in the middle of the profile sequence, it is ignored by the default CMM.

- If you specify a named color profile, it must be the first or the last profile. Otherwise, `CWConcatColorWorld` returns the value `cmCantConcatenateError`.

A after executing the `CWConcatColorWorld` function, you should call the function `CMCloseProfile` (page 728) for each profile to dispose of its reference.

Instead of passing in an array of profiles, you can specify a device link profile. For information on how to create a device link profile, see the `CWNewLinkProfile` function, which is described next.

Version Notes
The parameter description for `profileSet` includes changes in how this function is used starting with ColorSync version 2.5.

Note also that starting with version 2.5, use of the system profile has changed.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CWDisposeColorWorld

Releases the private storage associated with a color world when your application has finished using the color world.

```
void CWDisposeColorWorld (
   CMWorldRef cw
);
```

Parameters

cw

A color world reference of type `CMWorldRef` (page 942).

The function `NCWNewColorWorld` (page 846) and the function `CWConcatColorWorld` (page 823) both allocate color world references of type `CMWorldRef` (page 942).

Discussion
The following functions use color worlds. If you create a color world to pass to one of these functions, you must dispose of the color world when your application is finished with it.

- `CWMatchColors` (page 828)
- `CWCheckColors` (page 821)
- `CWMatchBitmap` (page 826)
- `CWCheckBitmap` (page 819)
- `CWMatchPixMap` (page 829)
- `CWCheckPixMap` (page 822)

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CWFillLookupTexture

Fills a 3-D lookup texture from a color world.

```
CMError CWFillLookupTexture (
   CMWorldRef cw,
   UInt32 gridPoints,
   UInt32 format,
   UInt32 dataSize,
   void *data
);
```

Parameters

cw
The color world to use.

gridPoints
The number of grid points per channel in the texture.

format
The format of pixels in texture; for example, `cmTextureRGBtoRGBX8`.

dataSize
The size in bytes of texture data to fill.

data
On output, points to the texture data to fill.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
You can use the resulting table in OpenGL to accelerate color management in hardware.

Availability
Available in Mac OS X v. 10.3 and later.

Declared In
`CMApplication.h`

CWMatchBitmap

Matches the colors of a bitmap to the gamut of a destination device using the profiles specified by a color world.

```
CMError CWMatchBitmap (
   CMWorldRef cw,
   CMBitmap *bitmap,
   CMBitmapCallBackUPP progressProc,
   void *refCon,
   CMBitmap *matchedBitmap
);
```

Parameters

cw

A reference to a color world of type `CMWorldRef` (page 942) in which matching is to occur.

The functions `NCWNewColorWorld` (page 846) and `CWConcatColorWorld` (page 823) both allocate color world references of type `CMWorldRef` (page 942).

bitmap

A pointer to a bitmap of type `CMBitmap` (page 882) whose colors are to be matched.

progressProc

A calling program–supplied universal procedure pointer to a callback function that allows your application to monitor progress or abort the operation as the bitmap colors are matched. The default CMM calls your function approximately every half-second unless color matching occurs in less time this happens when there is a small amount of data to be matched. If the function returns a result of `true`, the operation is aborted. To match colors without monitoring the process, specify `NULL` for this parameter. For a description of the function your application supplies, see the function `CMBitmapCallBackProcPtr` (page 852).

refCon

A pointer to a reference constant for application data passed through as a parameter to calls to the `progressProc` function.

matchedBitmap

A pointer to a bitmap. On return, contains the color-matched image. You must allocate the pixel buffer pointed to by the `image` field of the structure `CMBitmap` (page 882). If you specify `NULL` for `matchedBitMap`, then the source bitmap is matched in place.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `CWMatchBitmap` function matches a bitmap using the profiles specified by the given color world.

You should ensure that the buffer pointed to by the `image` field of the bitmap passed in the `bitMap` parameter is zeroed out before you call this function.

The ColorSync Manager does not explicitly support a CMY color space. However, for printers that have a CMY color space, you can use either of the following circumventions to make the adjustment:

- You can use a CMY profile, which the ColorSync Manager does support, with a CMYK color space. If you specify a CMYK color space in this case, the ColorSync Manager zeroes out the K channel to simulate a CMY color space.
- You can use an RGB color space and pass in the bitmap along with an RGB profile, then perform the conversion from RGB to CMY yourself.

For this function to execute successfully, the source profile's `dataColorSpace` field value and the `space` field value of the source bitmap pointed to by the `bitMap` parameter must specify the same data color space. Additionally, the destination profile's `dataColorSpace` field value and the `space` field value of the resulting bitmap pointed to by the `matchedBitMap` parameter must specify the same data color space, unless the destination profile is a named color space profile.

If you set `matchedBitMap` to `NULL` to specify in-place matching, you must be sure the space required by the destination bitmap is less than or equal to the size of the source bitmap.

Version Notes
The color spaces currently supported for the CWMatchBitmap function are defined in "Color Space Constants With Packing Formats" (page 962). Support for the following color space constants, was added with ColorSync version 2.5:

- `cmGray16Space`
- `cmGrayA32Space`
- `cmRGB48Space`.
- `cmCMYK64Space`
- `cmLAB48Space`

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CWMatchColors

Matches colors in a color list, using the specified color world.

```
CMError CWMatchColors (
   CMWorldRef cw,
   CMColor *myColors,
   size_t count
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 942) that describes how matching is to occur in the color-matching session.

The functions `NCWNewColorWorld` (page 846) and `CWConcatColorWorld` (page 823) both allocate color world references of type `CMWorldRef` (page 942).

myColors

A pointer to an array containing a list of colors of type `CMColor` (page 884). On input, contains the list of colors to match. On return, contains the list of matched colors specified in the color data space of the color world's destination profile.

count

A one-based count of the number of colors in the color list of the `myColors` array.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CWMatchColors` function matches colors according to the profiles corresponding to the specified color world. On input, the color values in the `myColors` array are assumed to be specified in the data color space of the source profile. On return, the color values in the `myColors` array are transformed to the data color space of the destination profile.

All color management modules (CMM)s must support this function.

This function supports color-matching sessions set up with one of the multichannel color data types.

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

CWMatchPixMap

Matches a pixel map in place based on a specified color world. (Deprecated in Mac OS X v10.4.)

```
CMError CWMatchPixMap (
   CMWorldRef cw,
   PixMap *myPixMap,
   CMBitmapCallBackUPP progressProc,
   void *refCon
);
```

Parameters

cw

A reference to the color world of type `CMWorldRef` (page 942) in which matching is to occur.

The functions `NCWNewColorWorld` (page 846) and `CWConcatColorWorld` (page 823) both allocate color world references of type `CMWorldRef` (page 942).

myPixMap

A pointer to the pixel map to match. A pixel map is a QuickDraw structure describing pixel data. The pixel map must be nonrelocatable; to ensure this, you should lock the handle to the pixel map before you call this function. See the QuickDraw Reference for a description of the `PixMap` data type.

progressProc

A function supplied by your application to monitor progress or abort the operation as the pixel map colors are matched. The default CMM calls your function approximately every half-second, unless matching is completed in less time.

If the function returns a result of `true`, the operation is aborted. You specify `NULL` for this parameter if your application will not monitor the pixel map color matching. For information on the callback function and its type definition, refer to the function `CMProfileFilterProcPtr` (page 864).

refCon

A pointer to a reference constant for application data that is passed as a parameter to calls to `progressProc`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `CWMatchPixMap` function matches a pixel map in place using the profiles specified by the given color world. The preferred CMM, as determined by the ColorSync Manager based on the color world configuration, is called to perform the color matching.

If the preferred CMM is not available, then the ColorSync Manager calls the default CMM to perform the matching. If the preferred CMM is available but it does not implement the `CMMatchPixMap` function, then the ColorSync Manager unpacks the colors in the pixel map to create a color list and calls the preferred CMM's `CMMatchColors` function, passing to this function the list of colors to match. Every CMM must support the `CMMatchColors` function.

For this function to execute successfully, the source and destination profiles' data color spaces (`dataColorSpace` field) must be RGB to match the data color space of the pixel map, which is implicitly RGB. For color spaces other than RGB, you should use the function `CWMatchBitmap` (page 826).

If you specify a pointer to a callback function in the `progressProc` parameter, the CMM performing the color matching calls your function to monitor progress of the session. Each time the CMM calls your function, it passes the function any data you specified in the `CWMatchPixMap` function's `refCon` parameter. If the ColorSync Manager performs the color matching, it calls your callback monitoring function once every scan line during this process.

You can use the reference constant to pass in any kind of data your callback function requires. For example, if your application uses a dialog box with a progress bar to inform the user of the color-matching session's progress, you can use the reference constant to pass the dialog box's window reference to the callback routine. For information about the callback function, see the function `CMBitmapCallBackProcPtr` (page 852).

Applications do not interact directly with the function `CWMatchColors` (page 828).

Availability
Available in CarbonLib 1.0 and later when ColorSync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CWNewLinkProfile

Creates a device link profile based on the specified set of profiles. (Deprecated in Mac OS X v10.5.)

```
CMError CWNewLinkProfile (
   CMProfileRef *prof,
   const CMProfileLocation *targetLocation,
   CMConcatProfileSet *profileSet
);
```

Parameters

prof

A pointer to an uninitialized profile reference of type `CMProfileRef` (page 925). On return, points to the new device link profile reference.

targetLocation

On return, a pointer to a location specification for the resulting profile. A device link profile cannot be a temporary profile: that is, it cannot have a location type of `cmNoProfileBase`.

profileSet

On input, a pointer to an array of profiles describing the processing to carry out. The array is in processing order—source through destination. For a description of the `CMConcatProfileSet` (page 887) data type, see `CMHeader` (page 898).

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

This discussion is accurate for versions of ColorSync prior to 2.5. See the version notes below for changes starting with version 2.5.

You can use this function to create a new single profile containing a set of profiles and pass the device link profile to the function `CWConcatColorWorld` (page 823) instead of specifying each profile in an array. A device link profile provides a means of storing in concatenated format a series of device profiles and non-device profiles that are used repeatedly in the same sequence.

The only way to use a device link profile is to pass it to the `CWConcatColorWorld` function as the sole profile specified by the array passed in the `profileSet` parameter.

The zero-based `keyIndex` field of the `CMConcatProfileSet` data structure specifies the index of the profile within the device link profile whose preferred CMM is used for the entire color-matching or color-checking session. The profile header's `CMMType` field specifies the preferred CMM for the specified profile. This CMM will fetch the profile elements necessary for the session.

The quality flag setting—indicating normal mode, draft mode, or best mode—specified by the first profile prevails for the entire session the quality flags of profiles that follow in the sequence are ignored. The quality flag setting is stored in the `flag` field of the profile header. See `CM2Header` (page 875) for more information on the use of flags.

The rendering intent specified by the first profile is used to color match to the second profile, the rendering intent specified by the second profile is used to color match to the third profile, and so on through the series of concatenated profiles.

The following rules govern the content and use of a device link profile:

- The first and last profiles you specify in the profiles array for a device link profile must be device profiles.
- You cannot specify a named color profile.
- You cannot include another device link profile in the series of profiles you specify in the profiles array.

- The only way to use a device link profile is to pass it to the `CWConcatColorWorld` function as the sole profile specified by the array passed in the `profileSet` parameter.
- You cannot embed a device link profile in an image.
- You cannot specify `NULL` to indicate the system profile.

This function privately maintains all the profile information required by the color world for color-matching and color-checking sessions. Therefore, after executing the `CWNewLinkProfile` function, you should call the `CMCloseProfile` (page 728) function for each profile used to build a device link profile (to dispose of each profile reference).

Version Notes
Note that starting with version 2.5, use of the system profile has changed.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

DisposeCMBitmapCallBackUPP

Disposes of a universal procedure pointer (UPP) to a bitmap callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMBitmapCallBackUPP (
   CMBitmapCallBackUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

DisposeCMConcatCallBackUPP

Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMConcatCallBackUPP (
   CMConcatCallBackUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CMTypes.h`

DisposeCMFlattenUPP

Disposes of a universal procedure pointer (UPP) to a data-flattening callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMFlattenUPP (
   CMFlattenUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CMTypes.h`

DisposeCMMIterateUPP

Disposes of a universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMMIterateUPP (
   CMMIterateUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
CMApplication.h

DisposeCMProfileAccessUPP

Disposes of a universal procedure pointer (UPP) to a profile-access callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMProfileAccessUPP (
   CMProfileAccessUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

DisposeCMProfileFilterUPP

Disposes of a universal procedure pointer (UPP) to a profile-filter callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMProfileFilterUPP (
   CMProfileFilterUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

DisposeCMProfileIterateUPP

Disposes of a universal procedure pointer (UPP) to a profile-iteration callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeCMProfileIterateUPP (
   CMProfileIterateUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
CMApplication.h

InvokeCMBitmapCallBackUPP

Invokes a universal procedure pointer (UPP) to a bitmap callback. (Deprecated in Mac OS X v10.5.)

```
Boolean InvokeCMBitmapCallBackUPP (
   SInt32 progress,
   void *refCon,
   CMBitmapCallBackUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMBitmapCallBackProcPtr" (page 852) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
CMTypes.h

InvokeCMConcatCallBackUPP

Invokes a universal procedure pointer (UPP) to a progress-monitoring callback. (Deprecated in Mac OS X v10.5.)

```
Boolean InvokeCMConcatCallBackUPP (
   SInt32 progress,
   void *refCon,
   CMConcatCallBackUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMConcatCallBackProcPtr" (page 853) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

InvokeCMFlattenUPP

Invokes a universal procedure pointer (UPP) to a data-flattening callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMFlattenUPP (
   SInt32 command,
   long *size,
   void *data,
   void *refCon,
   CMFlattenUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMFlattenProcPtr" (page 855) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

InvokeCMMIterateUPP

Invokes a universal procedure pointer (UPP) to a a progress-monitoring callback for the `CMIterateCMMInfo` function. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMMIterateUPP (
   CMMInfo *iterateData,
   void *refCon,
   CMMIterateUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMMIterateProcPtr" (page 862) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
CMApplication.h

InvokeCMProfileAccessUPP

Invokes a universal procedure pointer (UPP) to a profile-access callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMProfileAccessUPP (
   SInt32 command,
   SInt32 offset,
   SInt32 *size,
   void *data,
   void *refCon,
   CMProfileAccessUPP userUPP
);
```

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMProfileAccessProcPtr" (page 862) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

InvokeCMProfileFilterUPP

Invokes a universal procedure pointer (UPP) to a profile-filter callback. (Deprecated in Mac OS X v10.5.)

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMProfileFilterProcPtr" (page 864) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
CMTypes.h

InvokeCMProfileIterateUPP

Invokes a universal procedure pointer (UPP) to a profile-iteration callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeCMProfileIterateUPP (
   CMProfileIterateData *iterateData,
   void *refCon,
   CMProfileIterateUPP userUPP
);
```

Parameters

Return Value
A result code. See "ColorSync Manager Result Codes" (page 1020).

Discussion
In most cases, you do not need to call this function as ColorSync Manager invokes your callback for you. See the "CMProfileIterateProcPtr" (page 865) callback for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMApplication.h`

NCMBeginMatching

Sets up a QuickDraw-specific ColorSync matching session, using the specified source and destination profiles. (Deprecated in Mac OS X v10.4.)

```
CMError NCMBeginMatching (
   CMProfileRef src,
   CMProfileRef dst,
   CMMatchRef *myRef
);
```

Parameters

src

A profile reference of type `CMProfileRef` (page 925) that specifies the source profile for the matching session. Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 752) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 767) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

See the QuickDraw Reference for a description of the PixMap data type.

dst

A profile reference of type `CMProfileRef` (page 925) that specifies the destination profile for the matching session. Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 752) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 767) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.See the QuickDraw Reference for a description of the PixMap data type.

myRef

A pointer to a matching session. On return, it specifies the QuickDraw-specific matching session that was set up. See the QuickDraw Reference for a description of the PixMap data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `NCMBeginMatching` function sets up a QuickDraw-specific matching session, telling the ColorSync Manager to match all colors drawn to the current graphics device using the specified source and destination profiles.

The `NCMBeginMatching` function returns a reference to the color-matching session. You must later pass this reference to the function `CMEndMatching` (page 747) to conclude the session.

The source and destination profiles define how the match is to occur. Passing `NULL` for either the source or destination profile is equivalent to passing the system profile. If the current device is a screen device, matching to all screen devices occurs.

The `NCMBeginMatching` and `CMEndMatching` functions can be nested. In such cases, the ColorSync Manager matches to the most recently added profiles first. Therefore, if you want to use the `NCMBeginMatching`–`CMEndMatching` pair to perform a page preview—which typically entails color matching from a source device (scanner) to a destination device (printer) to a preview device (display)— you first call `NCMBeginMatching` with the printer-to-display profiles, and then call `NCMBeginMatching` with the scanner-to-printer profiles. The ColorSync Manager then matches all drawing from the scanner to the printer and then back to the display. The print preview process entails multiprofile transformations. The ColorSync Manager general purpose functions (which include the use of concatenated profiles well suited to print-preview processing) offer an easier and faster way to do this. These functions are described in "Matching Colors Using General Purpose Functions".

If you call `NCMBeginMatching` before drawing to the screen's graphics device (as opposed to an offscreen device), you must call `CMEndMatching` to finish a matching session before calling `WaitNextEvent` or any other routine (such as Window Manager routines) that could draw to the screen. Failing to do so will cause unwanted matching to occur. Furthermore, if a device has color matching enabled, you cannot call the `CopyBits` procedure to copy from it to itself unless the source and destination rectangles are the same.

Even if you call the `NCMBeginMatching` function before calling the QuickDraw `DrawPicture` function, the ColorSync picture comments such as `cmEnableMatching` and `cmDisableMatching` are not acknowledged. For the ColorSync Manager to recognize these comments and allow their use, you must call the function `NCMUseProfileComment` (page 843) for color matching using picture comments.

This function causes matching for the specified devices rather than for the current color graphics port.

The `NCMBeginMatching` function uses QuickDraw and performs color matching in a manner acceptable to most applications. However, if your application needs a finer level of control over color matching, it can use the general purpose functions described in "Matching Colors Using General Purpose Functions".

Version Notes
The parameter descriptions for `src` and `dst` describe changes in how this function is used starting with ColorSync version 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NCMDrawMatchedPicture

Matches a picture's colors to a destination device's color gamut, as the picture is drawn, using the specified destination profile. (Deprecated in Mac OS X v10.4.)

```
void NCMDrawMatchedPicture (
   PicHandle myPicture,
   CMProfileRef dst,
   Rect *myRect
);
```

Parameters

myPicture

The QuickDraw picture whose colors are to be matched. See the QuickDraw Reference for a description of the PixMap data type.

dst

A profile reference of type `CMProfileRef` (page 925) to the profile of the destination device. Starting with ColorSync version 2.5, if you know the destination display device, you can call `CMGetProfileByAVID` (page 767) to get the specific profile for the display, or you can call `CMGetDefaultProfileBySpace` (page 752) to get the default profile for the RGB color space,.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

See the QuickDraw Reference for a description of the PixMap data type.

myRect

A pointer to a destination rectangle for rendering the picture specified by `myPicture`.

Return Value
This function does not return an error value. Instead, after calling `NCMDrawMatchedPicture` you call the `QDError` routine to determine if an error has occurred.

Discussion
The `NCMDrawMatchedPicture` function operates in the context of the current color graphics port. This function sets up and takes down a color-matching session. It automatically matches all colors in a picture to the destination profile for a destination device as the picture is drawn. It uses the ColorSync system profile as the initial source profile and any embedded profiles as they are encountered thereafter. (Because color-matching picture comments embedded in the picture to be matched are recognized, embedded profiles are used.)

The ColorSync Manager defines five picture comment kinds, as described in "Picture Comment Kinds" (page 995). For embedding to work correctly, each embedded profile that is used for matching must be terminated by a picture comment of kind `cmEndProfile`. If a picture comment is not specified to end the profile after drawing operations using that profile are performed, the profile will remain in effect until another embedded profile is introduced that has a picture comment kind of `cmBeginProfile`. To avoid unexpected matching effects, always pair use of the `cmBeginProfile` and `cmEndProfile` picture comments. When the ColorSync Manager encounters a `cmEndProfile` picture comment, it restores use of the system profile for matching until it encounters another `cmBeginProfile` picture comment.

The picture is drawn with matched colors to all screen graphics devices. If the current graphics device is not a screen device, matching occurs for that graphics device only.

If the current port is not a color graphics port, then calling this function is equivalent to calling `DrawPicture`, in which case no color matching occurs.

Version Notes
The parameter description for `dst` describes changes in how this function is used starting with ColorSync version 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NCMGetProfileLocation

Obtains either a profile location structure for a specified profile or the size of the location structure for the profile.

```
CMError NCMGetProfileLocation (
   CMProfileRef prof,
   CMProfileLocation *theProfile,
   UInt32 *locationSize
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925). Before calling `NCMGetProfileLocation`, you set the reference to specify the profile for which you wish to obtain the location or location structure size.

theProfile

A pointer to a profile location structure, as described in `CMProfileLocation` (page 924). If you pass `NULL`, `NCMGetProfileLocation` returns the size of the profile location structure for the profile specified by `prof` in the `locationSize` parameter. If you instead pass a pointer to memory you have allocated for the structure, on return, the structure specifies the location of the profile specified by `prof`.

locationSize

A pointer to a value of type `long`. If you pass `NULL` for the `profLoc` parameter, on return, `locationSize` contains the size in bytes of the profile location structure for the profile specified by `prof`. If you pass a pointer to a profile location structure in `profLoc`, set `locationSize` to the size of the structure before calling `NCMGetProfileLocation`, using the constant `cmCurrentProfileLocationSize`.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The `NCMGetProfileLocation` function is available starting with ColorSync version 2.5. It differs from its predecessor, `CMGetProfileLocation` (page 770), in that the newer version has a parameter for the size of the location structure for the specified profile.

You should use `NCMGetProfileLocation` rather than `CMGetProfileLocation` for the following reasons:

- Code using the older version (`CMGetProfileLocation`) may not be as easily ported to other platforms.
- Specifying the size of the profile location structure ensures that it can grow, if necessary, in the future.

The best way to use `NCMGetProfileLocation` is to call it twice:

1. Pass a reference to the profile to locate in the `prof` parameter and `NULL` for the `profLoc` parameter. `NCMGetProfileLocation` returns the size of the location structure in the `locationSize` parameter.
2. Allocate enough space for a structure of the returned size, then call the function again, passing a pointer in the `profLoc` parameter; on return, the structure specifies the location of the profile.

It is possible to call `NCMGetProfileLocation` just once, using the constant `cmCurrentProfileLocationSize` for the size of the allocated profile location structure and passing the same constant for the `locationSize` parameter. The constant `cmCurrentProfileLocationSize` may change in the future, but will be consistent within the set of headers you build your application with. However, if the size of the `CMProfileLocation` structure changes in a future version of ColorSync (and the value of `cmCurrentProfileLocationSize` as well) and you do not rebuild your application, `NCMGetProfileLocation` may return an error.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

NCMSetSystemProfile

Sets the location of a color profile. (Deprecated in Mac OS X v10.5.)

```
CMError NCMSetSystemProfile (
   const CMProfileLocation *profLoc
);
```

Parameters

profLoc
The location of the profile. Commonly a profile is disk-file based. However, the profile may be a file-based profile, a handle-based profile, or a pointer-based profile.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
Prior to ColorSync 2.6, the function for setting the system profile supported only the `FSSpec` file specification type as a way of specifying a profile. This function allows for greater flexibility when specifying a system profile.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

NCMUnflattenProfile

Unflattens a previouslyflattened profile. (Deprecated in Mac OS X v10.5.)

```
CMError NCMUnflattenProfile (
   CMProfileLocation *targetLocation,
   CMFlattenUPP proc,
   void *refCon,
   Boolean *preferredCMMnotfound
);
```

Parameters

targetLocation
The location of the profile you want to unflatten. Commonly a profile is disk-file based. However, the profile may be a file-based profile, a handle-based profile, or a pointer-based profile.

proc
A user-defined procedure which is called during the unflatten operation.

refCon
A reference constant containing data specified by the calling application program.

preferredCMMnotfound
A return value indicating whether or not the CMM specified in the profile was found.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

NCMUseProfileComment

Automatically embeds a profile or a profile identifier into an open picture. (Deprecated in Mac OS X v10.4.)

```
CMError NCMUseProfileComment (
   CMProfileRef prof,
   UInt32 flags
);
```

Parameters

prof

A profile reference of type `CMProfileRef` (page 925) to the profile to embed. See the QuickDraw Reference for a description of the PixMap data type.

flags

A flag value in which individual bits determine settings. "Embedded Profile Identifiers" (page 982) describes constants for use with this parameter. For example, you pass `cmEmbedWholeProfile` to embed a whole profile or `cmEmbedProfileIdentifier` to embed a profile identifier. No other values are currently defined; all other bits are reserved for future use.

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion

The `NCMUseProfileComment` function automatically generates the picture comments required to embed the specified profile or profile identifier into the open picture.

To embed a profile, you use the constant `cmEmbedWholeProfile` to set the `flags` parameter before calling `NCMUseProfileComment`. The `NCMUseProfileComment` function calls the QuickDraw `PicComment` function with a picture comment `kind` value of `cmComment` and a 4-byte selector that describes the type of data in the picture comment: `cmBeginProfileSel` to begin the profile, `cmContinueProfileSel` to continue, and `cmEndProfileSel` to end the profile. These constants are described in "Picture Comment Selectors" (page 997).

If the size in bytes of the profile and the 4-byte selector together exceed 32 KB, this function segments the profile data and embeds the multiple segments in consecutive order using selector `cmContinueProfileSel` to embed each segment.

To embed a profile identifier of type `CMProfileIdentifier` (page 921) , you use the constant `cmEmbedProfileIdentifier` to set the `flags` parameter before calling `NCMUseProfileComment`. The function extracts the necessary information from the profile reference (`prof`) to embed a profile identifier for the profile. The profile reference can refer to a previously embedded profile, or to a profile on disk in the ColorSync Profiles folder.

You can use this function to embed most types of profiles in an image, including device link profiles, but not abstract profiles. You cannot use this function to embed ColorSync 1.0 profiles in an image.

The `NCMUseProfileComment` function precedes the profile it embeds with a picture comment of kind `cmBeginProfile`. For embedding to work correctly, the currently effective profile must be terminated by a picture comment of kind `cmEndProfile` after drawing operations using that profile are performed. You are responsible for adding the picture comment of kind `cmEndProfile`. If a picture comment was not specified to end the profile following the drawing operations to which the profile applies, the profile will remain in effect until the next embedded profile is introduced with a picture comment of kind `cmBeginProfile`. However, use of the next profile might not be the intended action. Always pair use of the `cmBeginProfile` and `cmEndProfile` picture comments. When the ColorSync Manager encounters a `cmEndProfile` picture comment, it restores use of the system profile for matching until it encounters another `cmBeginProfile` picture comment.

Version Notes

In ColorSync 2.0, the `flags` parameter was ignored and the routine always embedded the entire profile.

In ColorSync 2.0, if the `prof` parameter refers to a version 1.0 profile, the profile is not embedded into the picture correctly. In ColorSync versions starting with 2.1, this bug has been fixed. One possible workaround for this problem in ColorSync 2.0 is to call `CMCopyProfile` to copy the 1.0 profile reference into a handle. The handle can then be embedded into the picture using `CMUseProfileComment`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NCWConcatColorWorld

Defines a color world for color transformations among a series of concatenated profiles.

```
CMError NCWConcatColorWorld (
   CMWorldRef *cw,
   NCMConcatProfileSet *profileSet,
   CMConcatCallBackUPP proc,
   void *refCon
);
```

Parameters

cw
A reference to a color world that the ColorSync Manager returns if the function completes successfully. You pass this reference to other functions that use the color world for color-matching and color-checking sessions.

profileSet
An array of profiles describing the processing to be carried out. The array is in processing order source through destination.

proc
A calling-program-supplied callback function that allows your application to monitor progress or abort the operation.

refCon
A reference constant containing data specified by the calling application program.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The caller can override the color management module (CMM) that would normally be selected by ColorSync by providing a CMM identifier in the `NCMConcatProfileSet` structure, or pass 0 to accept ColorSync's CMM selection (note that this could either be the user's preferred CMM selection or the CMM called for in the profile). The *flags* and k parameters are provided to allow easy customization of such attributes as quality and gamut-checking, while preserving the other settings. Each profile in the set can be customized by overriding the intent, and the selection of the transform tag. Together with other profiles, a custom-rendering environment can be set up to transform to or from device-dependent spaces with a minimum of gamut compression and/or unnecessary transformations to and from connection spaces. This flexibility comes at the price of specific knowledge of the profile contents and how device gamuts overlap.

Note that for standard input and output device profiles, A2B and B2A tags represent transforms from data space to connection space and from connection space to data space, respectively. Under these circumstances, the caller would not normally be able to use the same transform tags (e.g., `kUseAtoB`) consecutively, since a connection space would not be the same as the subsequent data space. If the spaces aren't the same, the caller will get `a cmCantConcatenateError` error returned. For profiles of type `cmLinkClass`, `cmAbstractClass`, `cmColorSpaceClass` , and `cmNamedColorClass` , these constants are not always meaningful, and the caller is encouraged to think in terms of the actual tags present in the profiles (e.g., A2B0 or B2A0). Under these conditions, it may well be appropriate to specify two transform tags of the same type consecutively, as long as the actual color spaces align in between tags. If this is not the case, a `cmCantConcatenateError` error is returned.

The callback proc is provided as protection against the appearance of a stalled machine during lengthy color world processing. If a CMM takes more than several seconds to process the information and create a color world, it will call the callback proc, if one is provided, and pass it the `refCon` provided. This is also true for `NCWNewLinkProfile`.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

NCWNewColorWorld

Creates a color world for color matching based on the specified source and destination profiles.

```
CMError NCWNewColorWorld (
   CMWorldRef *cw,
   CMProfileRef src,
   CMProfileRef dst
);
```

Parameters

cw

A pointer to a color world. On return, a reference to a matching session color world of type `CMWorldRef` (page 942). You pass this reference to other functions that use the color world.

src

A profile reference of type `CMProfileRef` (page 925) that specifies the source profile for the color-matching world. This profile's `dataColorSpace` element corresponds to the source data type for subsequent calls to functions that use this color world.

Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 752) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 767) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

dst

A profile reference of type `CMProfileRef` (page 925) that specifies the destination profile for the color-matching world. This profile's `dataColorSpace` element corresponds to the destination data type for subsequent calls to functions using this color world.

Starting with ColorSync version 2.5, you can call `CMGetDefaultProfileBySpace` (page 752) to get the default profile for a specific color space or `CMGetProfileByAVID` (page 767) to get a profile for a specific display.

With any version of ColorSync, you can specify a `NULL` value to indicate the ColorSync system profile. Note, however, that starting with version 2.5, use of the system profile has changed.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
You must set up a color world before your application can perform general purpose color-matching or color-checking operations. To set up a color world for these operations, your application can call `NCWNewColorWorld` after obtaining references to the profiles to use as the source and destination profiles for the color world. The following rules govern the types of profiles allowed:

- You can specify a device profile or a color space conversion profile for the source and destination profiles.
- You can not specify a device link profile or an abstract profile for either the source profile or the destination profile.
- Only one profile can be a named color profile.
- You can specify the system profile explicitly by reference or by giving `NULL` for either the source profile or the destination profile.

You should call the function `CMCloseProfile` (page 728) for both the source and destination profiles to dispose of their references after execution of the `NCWNewColorWorld` function.

The quality flag setting (indicating normal mode, draft mode, or best mode) specified by the source profile prevails for the entire session. The quality flag setting is stored in the `flags` field of the profile header. See `CM2Header` (page 875) and "Flag Mask Definitions for Version 2.x Profiles" (page 983) for more information on the use of flags. The rendering intent specified by the source profile also prevails for the entire session.

The function `CWConcatColorWorld` (page 823) also allocates a color world reference of type `CMWorldRef` (page 942).

Version Notes
The parameter descriptions for `src` and `dst` describe changes in how this functions is used starting with ColorSync version 2.5.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

NCWNewLinkProfile

Obtains a profile reference for the specified by the profile location.

```
CMError NCWNewLinkProfile (
   CMProfileRef *prof,
   const CMProfileLocation *targetLocation,
   NCMConcatProfileSet *profileSet,
   CMConcatCallBackUPP proc,
   void *refCon
);
```

Parameters

prof
The returned profile reference.

targetLocation
The location of the profile. Commonly a profile is disk-file based. However, the profile may be a file-based profile, a handle-based profile, or a pointer-based profile.

profileSet
A pointer to the profile set structure.

proc
A calling-program-supplied callback function that allows your application to monitor progress or abort the operation.

refCon
A reference constant containing data specified by the calling application program.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Discussion
The same new flexibility in creating color worlds is extended to link profiles, which are not assumed to go from input device color space to output device color space. The returned profile is open, and should be closed when you are finished with it.

Availability
Available in CarbonLib 1.0 and later when ColorSync 2.6 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`CMApplication.h`

NewCMBitmapCallBackUPP

Creates a new universal procedure pointer (UPP) to a bitmap callback. (Deprecated in Mac OS X v10.5.)

```
CMBitmapCallBackUPP NewCMBitmapCallBackUPP (
   CMBitmapCallBackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your bitmap callback function.

Return Value
The universal procedure pointer.

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

NewCMConcatCallBackUPP

Creates a new universal procedure pointer (UPP) to a progress-monitoring callback. (Deprecated in Mac OS X v10.5.)

```
CMConcatCallBackUPP NewCMConcatCallBackUPP (
   CMConcatCallBackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your progress-monitoring callback function.

Return Value
The universal procedure pointer.

Discussion
The callback protects against the appearance of a stalled machine during lengthy color world processing. If a CMM takes more than several seconds to process the information and create a color world, it will call the callback, if one is provided, and pass it the `refCon` provided. Passed to the functions `NCWNewLinkProfile` or `NCWConcatColorWorld` function .

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`CMTypes.h`

NewCMFlattenUPP

Creates a new universal procedure pointer (UPP) to a data-flattening callback. (Deprecated in Mac OS X v10.5.)

```
CMFlattenUPP NewCMFlattenUPP (
   CMFlattenProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your data-flattening callback function.

Return Value

The universal procedure pointer.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CMTypes.h`

NewCMMIterateUPP

Creates a new universal procedure pointer (UPP) to a progress-monitoring callback for the `CMIterateCMMInfo` function. (Deprecated in Mac OS X v10.5.)

```
CMMIterateUPP NewCMMIterateUPP (
   CMMIterateProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your progress-monitoring callback function.

Return Value

The universal procedure pointer.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`CMApplication.h`

NewCMProfileAccessUPP

Creates a new universal procedure pointer (UPP) to a profile-access callback. (Deprecated in Mac OS X v10.5.)

```
CMProfileAccessUPP NewCMProfileAccessUPP (
   CMProfileAccessProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your profile-access callback function.

Return Value
The universal procedure pointer.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMTypes.h`

NewCMProfileFilterUPP

Creates a new universal procedure pointer (UPP) to a profile-filter callback. (Deprecated in Mac OS X v10.5.)

```
CMProfileFilterUPP NewCMProfileFilterUPP (
   CMProfileFilterProcPtr userRoutine
);
```

Parameters
userRoutine
A pointer to your profile-filter callback function.

Return Value
The universal procedure pointer.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CMTypes.h`

NewCMProfileIterateUPP

Creates a new universal procedure pointer (UPP) to a profile-iteration callback. (Deprecated in Mac OS X v10.5.)

```
CMProfileIterateUPP NewCMProfileIterateUPP (
   CMProfileIterateProcPtr userRoutine
);
```

Parameters
userRoutine
A pointer to your profile-iteration callback function.

Return Value
The universal procedure pointer.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`CMApplication.h`

Callbacks

CMBitmapCallBackProcPtr

Defines a pointer to a bitmap callback function that function reports on the progress of a color-matching or color-checking session being performed for a bitmap or a pixel map.

```
typedef Boolean (*MyCMBitmapCallBackProc)
(
    SInt16 progress,
    void * refCon
);
);
```

If you name your function `MyCMBitmapCallBackProc`, you would declare it like this:

```
Boolean MyCMBitmapCallBackProc (
    SInt16 progress,
    void * refCon
);
```

Parameters

progress

A byte count that begins at an arbitrary value when the function is first called. On each subsequent call, the value is decremented by an amount that can vary from call to call, but that reflects how much of the matching process has completed since the previous call. If the function is called at all, it will be called a final time with a byte count of 0 when the matching is complete.

refCon

The pointer to a reference constant passed to your `MyCMBitmapCallBack` function each time the color management module (CMM) calls your function.

Return Value
`False` indicates the color-matching or color-checking session should continue. `True` indicates the session should be aborted—for example, the user may be holding down the Command–period keys.

Discussion
Your `MyCMBitmapCallBack` function allows your application to monitor the progress of a color-matching or color-checking session for a bitmap or a pixel map. Your function can also terminate the matching or checking operation.

Your callback function is called by the CMM performing the matching or checking process if your application passes a pointer to your callback function in the `progressProc` parameter when it calls one of the following functions: `CWCheckBitmap` (page 819) , `CWMatchBitmap` (page 826) , `CWCheckPixMap` (page 822) , and `CWMatchPixMap` (page 829). Note that your callback function may not be called at all if the operation completes in a very short period.

The CMM used for the color-matching session calls your function at regular intervals. For example, the default CMM calls your function approximately every half-second unless the color matching or checking occurs in less time; this happens when there is a small amount of data to match or check.

Each time the ColorSync Manager calls your function, it passes to the function any data stored in the reference constant. This is the data that your application specified in the `refCon` parameter when it called one of the color-matching or checking functions.

For large bitmaps and pixel maps, your application can display a progress bar or other indicator to show how much of the operation has been completed. You might, for example, use the reference constant to pass to the callback function a window reference to a dialog box. You obtain information on how much of the operation has completed from the `progress` parameter. The first time your callback is called, this parameter contains an arbitrary byte count. On each subsequent call, the value is decremented by an amount that can vary from call to call, but that reflects how much of the matching process has completed since the previous call. Using the current value and the original value, you can determine the percentage that has completed. If the callback function is called at all, it will be called a final time with a byte count of 0 when the matching is complete.

To terminate the matching or checking operation, your function should return a value of `true`. Because pixel-map matching is done in place, an application that allows the user to terminate the process should revert to the prematched image to avoid partial mapping.

For bitmap matching, if the `matchedBitMap` parameter of the `CWMatchBitmap` function specifies `NULL`, to indicate that the source bitmap is to be matched in place, and the application allows the user to abort the process, you should also revert to the prematched bitmap if the user terminates the operation.

Each time the ColorSync Manager calls your progress function, it passes a byte count in the `progress` parameter. The last time the ColorSync Manager calls your progress function, it passes a byte count of 0 to indicate the completion of the matching or checking process. You should use the 0 byte count as a signal to perform any cleanup operations your function requires, such as filling the progress bar to completion to indicate to the user the end of the checking or matching session, and then removing the dialog box used for the display.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMConcatCallBackProcPtr

Defines a pointer to a progress-monitoring function that the ColorSync Manager calls during lengthy color world processing.

```
typedef Boolean (*CMConcatCallBackProcPtr)
(
    SInt32 progress,
    void *refCon
);
```

If you name your function `MyCMConcatCallBackProc`, you would declare it like this:

```
Boolean MyCMConcatCallBackProc (
    SInt32 progress,
    void *refCon
);
```

Parameters

progress

refCon

Discussion
If a CMM takes more than several seconds to process the information and create a color world, it will call the Callback proc, if one is provided, and pass it the refCon provided

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMCountImageProfilesProcPtr

Defines a pointer to a function that obtains a count of the number of embeded profiles for a given image..

```
typedef CMError (*CMCountImageProfilesProcPtr)
(
    const FSSpec * spec,
    UInt32 * count
);
```

If you name your function `MyCMCountImageProfilesProc`, you would declare it like this:

```
CMError MyCMCountImageProfilesProc (
    const FSSpec * spec,
    UInt32 * count
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

count

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMEmbedImageProcPtr

Defines a pointer to a function that embeds an image with an ICC profile..

```
typedef CMError (*CMEmbedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    CMProfileRef embProf
);
```

If you name your function `MyCMEmbedImageProc`, you would declare it like this:

```
CMError MyCMEmbedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    CMProfileRef embProf
);
```

Parameters

specFrom
See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

repl

embProf

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMFlattenProcPtr

Defines a pointer to a data transfer callback function that transfers profile data from the format for embedded profiles to disk file format or vice versa.

```
typedef OSErr (*CMFlattenProcPtr) (
    SInt32 command,
    SInt32 *size,
    void *data,
    void *refCon
);
```

If you name your function `MyCMFlattenProc`, you would declare it like this:

```
OSErr MyCMFlattenProc (
    SInt32 command,
    SInt32 *size,
    void *data,
    void *refCon
);
```

Parameters

command

The command with which the `MyCMFlattenCallback` function is called. This command specifies the operation the function is to perform.

size

A pointer to a size value. On input, the size in bytes of the data to transfer. On return, the size of the data actually transferred.

data

A pointer to the buffer supplied by the ColorSync Manager to use for the data transfer.

refCon

A pointer to a reference constant that holds the application data passed in from the functions `CMFlattenProfile` (page 748), `NCMUnflattenProfile` (page 843), `CMGetPS2ColorRenderingVMSize` (page 775), `CMGetPS2ColorRenderingIntent` (page 774), or `CMFlattenProfile` (page 748). Each time the CMM calls your `MyCMFlattenCallback` function, it passes this data to the function.

Starting in ColorSync version 2.5, the ColorSync Manager calls your function directly, without going through the preferred, or any, CMM.

Return Value

A result code. See "ColorSync Manager Result Codes" (page 1020).

Discussion

IThis callback can be used, for example, by PostScript functions to transfer data from a profile to text format usable by a PostScript driver. Starting in ColorSync version 2.5, the ColorSync Manager calls your data transfer function directly, without going through the preferred, or any, CMM. So any references to the CMM in the discussion that follows are applicable only to versions of ColorSync prior to version 2.5. Where the discussion does not involve CMMs, it is applicable to all versions of ColorSync.

Your `MyCMFlattenCallback` function is called to flatten and unflatten profiles or to transfer PostScript-related data from a profile to the PostScript format to send to an application or device driver.

The ColorSync Manager and the CMM communicate with the `MyCMFlattenCallback` function using the command parameter to identify the operation to perform. To read and write profile data, your function must support the following commands: `cmOpenReadSpool`, `cmOpenWriteSpool`, `cmReadSpool`, `cmWriteSpool`, and `cmCloseSpool`.

You determine the behavior of your `MyCMFlattenCallback` function. The following sections describe how your function might handle the flattening and unflattening processes.

Flattening a Profile:

The ColorSync Manager calls the specified profile's preferred CMM when an application calls the `CMFlattenProfile` function to transfer profile data embedded in a graphics document.

The ColorSync Manager determines if the CMM supports the `CMFlattenProfile` function. If so, the ColorSync Manager dispatches the `CMFlattenProfile` function to the CMM. If not, ColorSync calls the default CMM, dispatching the `CMFlattenProfile` function to it.

The CMM communicates with the `MyCMFlattenCallback` function using a command parameter to identify the operation to perform. The CMM calls your function as often as necessary, passing to it on each call any data transferred to the CMM from the `CMFlattenProfile` function's `refCon` parameter.

The ColorSync Manager calls your function with the following sequence of commands: `cmOpenWriteSpool`, `cmWriteSpool`, and `cmCloseSpool`. Here is how you should handle these commands:

- When the CMM calls your function with the `cmOpenWriteSpool` command, you should perform any initialization required to write profile data you receive from the CMM to a buffer or file.
- The CMM will call your function with the `cmWriteSpool` command as many times as necessary to transfer all the profile data to you. Each time you are called, you should receive the data and write it to your buffer or file, returning in the `size` parameter the number of bytes of data you actually accepted.
- When the CMM calls your function with the `cmCloseSpool` command, you should perform any required cleanup processes.

As part of this process, your function can embed the profile data in a graphics document, for example, a PICT file or a TIFF file. For example, your `MyCMFlattenCallback` function can call the QuickDraw `PicComment` function to embed the flattened profile in a picture.

Unflattening a Profile:

When an application calls the `CMUnflattenProfile` function to transfer a profile that is embedded in a graphics document to an independent disk file, the ColorSync Manager calls your `MyCMFlattenCallback` function with the following sequence of commands: `cmOpenReadSpool`, `cmReadSpool`, `cmCloseSpool`. Here is how you should handle these commands:

- When the ColorSync Manager calls your function with the `cmOpenReadSpool` command, you should perform any initialization required to read from the embedded profile format.
- The ColorSync Manager calls your function with the `cmReadSpool` command as many times as necessary, directing your function to extract the profile data from the embedded format in the image file and return it to the ColorSync Manager in the `data` buffer. For each call, the ColorSync Manager specifies in the `size` parameter the number of bytes of data you should return. Each time your function is called it should read and return the requested data; it should also specify in the `size` parameter the actual number of bytes of data it returns.
- When the ColorSync Manager calls your function with the `cmCloseSpool` command, you should perform any required cleanup processes.

Version Notes
Starting in ColorSync version 2.5, the ColorSync Manager calls your function directly, without going through the preferred, or any, CMM.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMGetImageSpaceProcPtr

Defines a pointer to a function that obtains the signature of the data color space in which the color values of colors in an image are expressed.

```
typedef CMError (*CMGetImageSpaceProcPtr)
(
    const FSSpec * spec,
    OSType * space
);
```

If you name your function `MyCMGetImageSpaceProc`, you would declare it like this:

```
CMError MyCMGetImageSpaceProc (
    const FSSpec * spec,
    OSType * space
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

space

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMGetIndImageProfileProcPtr

Defines a pointer to a function that obtains a specific embeded profile for a given image.

```
typedef CMError (*CMGetIndImageProfileProcPtr)
(
    const FSSpec * spec,
    UInt32 index,
    CMProfileRef * prof
);
```

If you name your function `MyCMGetIndImageProfileProc`, you would declare it like this:

```
CMError MyCMGetIndImageProfileProc (
    const FSSpec * spec,
    UInt32 index,
```

```
    CMProfileRef * prof
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

index

prof

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMIterateDeviceInfoProcPtr

Defines a pointer to a function that iterates through device information available on the system.

```
typedef OSErr (*CMIterateDeviceInfoProcPtr)
(
    const CMDeviceInfo * deviceInfo,
    void * refCon
);
```

If you name your function `MyCMIterateDeviceInfoProc`, you would declare it like this:

```
OSErr MyCMIterateDeviceInfoProc (
    const CMDeviceInfo * deviceInfo,
    void * refCon
);
```

Parameters

deviceData

refCon

Return Value
An `OSErr` value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMDeviceIntegration.h`

CMIterateDeviceProfileProcPtr

Defines a pointer to a function that iterates through the device profiles available on the system.

```
typedef OSErr (*CMIterateDeviceProfileProcPtr)
(
    const CMDeviceInfo * deviceInfo,
    const NCMDeviceProfileInfo * profileInfo,
    void * refCon
);
```

If you name your function `MyCMIterateDeviceProfileProc`, you would declare it like this:

```
OSErr MyCMIterateDeviceProfileProc (
    const CMDeviceInfo * deviceInfo,
    const NCMDeviceProfileInfo * profileInfo,
    void * refCon
);
```

Parameters

deviceData

profileData

refCon

Return Value
An `OSErr` value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMDeviceIntegration.h`

CMLinkImageProcPtr

Defines a pointer to a function that matches an image file with a device link profile.

```
typedef CMError (*CMLinkImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef lnkProf,
    UInt32 lnkIntent
);
```

If you name your function `MyCMLinkImageProc`, you would declare it like this:

```
CMError MyCMLinkImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef lnkProf,
    UInt32 lnkIntent
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl

qual

lnkProf

lnkIntent

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMMatchImageProcPtr

Defines a pointer to a function that color matches an image file.

```
typedef CMError (*CMMatchImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf
);
```

If you name your function `MyCMMatchImageProc`, you would declare it like this:

```
CMError MyCMMatchImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl
qual
srcProf
srcIntent
dstProf

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMMIterateProcPtr

Defines a pointer to a function that iterates through color management modules installed on the system.

```
typedef OSErr (*CMMIterateProcPtr) (
    CMMInfo * iterateData,
    void * refCon
);
```

If you name your function `MyCMMIterateProc`, you would declare it like this:

```
OSErr MyCMMIterateProc (
    CMMInfo * iterateData,
    void * refCon
);
```

Parameters
iterateData
refCon

Return Value
An `OSErr` value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMProfileAccessProcPtr

Defines a pointer to a profile access callback function that provides procedure-based access to a profile.

```
typedef OSErr (*CMProfileAccessProcPtr)
(
    SInt32 command,
    SInt32 offset,
    SInt32 *size,
    void *data,
    void *refCon
);
```

If you name your function `MyCMProfileAccessProc`, you would declare it like this:

```
OSErr MyCMProfileAccessProc (
    SInt32 command,
    SInt32 offset,
    SInt32 *size,
    void *data,
    void *refCon
);
```

Parameters

command

A command value indicating the operation to perform. Operation constants are described in "Profile Access Procedures" (page 998).

offset

For read and write operations, the offset from the beginning of the profile at which to read or write data.

size

A pointer to a size value. On input, for the `cmReadAccess` and `cmWriteAccess` command constants, a pointer to a value indicating the number of bytes to read or write; for the `cmOpenWriteAccess` command, the total size of the profile. On return, after reading or writing, the actual number of bytes read or written.

data

A pointer to a buffer containing data to read or write. On return, for a read operation, contains the data that was read.

refCon

A reference constant pointer that can store private data for the `CMProfileAccessCallback` function.

Return Value

An `OSErr` value.

Discussion

When your application calls the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, or `CMNewLinkProfile` functions, it may supply the ColorSync Manager with a profile location structure of type `CMProfileLocation` (page 924) that specifies a procedure that provides access to a profile. In the structure, you provide a universal procedure pointer to a profile access procedure supplied by you and, optionally, a pointer to data your procedure can use. The ColorSync Manager calls your procedure when the profile is created, initialized, opened, read, updated, or closed.

When the ColorSync Manager calls your profile access procedure, it passes a constant indicating the operation to perform. The operations include creating a new profile, reading from the profile, writing the profile, and so on. Operation constants are described in "Profile Access Procedures" (page 998). Your procedure must be able to respond to each of these constants.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
CMTypes.h

CMProfileFilterProcPtr

Defines a pointer to a profile filter callback function that examines the profile whose reference you specify and determines whether to include it in the profile search result list.

```
typedef Boolean (*CMProfileFilterProcPtr)
(
    CMProfileRef prof,
    void * refCon
);
```

If you name your function MyCMProfileFilterProc, you would declare it like this:

```
Boolean MyCMProfileFilterProc (
    CMProfileRef prof,
    void * refCon
);
```

Parameters

prof

A profile reference of type CMProfileRef (page 925) to the profile to test.

refCon

A pointer to a reference constant that holds data passed through from the CMNewProfileSearch function or the CMUpdateProfileSearch function.

Return Value
A value of false indicates that the profile should be included; true indicates that the profile should be filtered out.

Discussion
Your MyCMProfileFilterCallback function is called after the CMNewProfileSearch function searches for profiles based on the search record's contents as specified by the search bitmask.

When your application calls CMNewProfileSearch, it passes a reference to a search specification record of type CMSearchRecord of type CMSearchRecord (page 932) that contains a filter field. If the filter field contains a pointer to your MyCMProfileFilterCallback function, then your function is called to determine whether to exclude a profile from the search result list. Your function should return true for a given profile to exclude that profile from the search result list. If you do not want to filter profiles beyond the criteria in the search record, specify a NULL value for the search record's filter field.

After a profile has been included in the profile search result based on criteria specified in the search record, your MyCMProfileFilterCallback function can further examine the profile. For example, you may wish to include or exclude the profile based on criteria such as an element or elements not included in the CMSearchRecord search record. Your MyCMProfileFilterCallback function can also perform searching using AND or OR logic.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
CMTypes.h

CMProfileIterateProcPtr

Defines a pointer to a profile iteration callback function that the ColorSync Manager calls for each found profile file as it iterates over the available profiles.

```
typedef OSErr (*CMProfileIterateProcPtr)
(
    CMProfileIterateData * iterateData,
    void * refCon
);
```

If you name your function `MyCMProfileIterateProc`, you would declare it like this:

```
OSErr MyCMProfileIterateProc (
    CMProfileIterateData * iterateData,
    void * refCon
);
```

Parameters

iterateData

A pointer to a structure of type `CMProfileIterateData` (page 923). When the function `CMIterateColorSyncFolder` (page 780) calls `MyProfileIterateCallback`, as it does once for each found profile, the structure contains key information about the profile.

refCon

An untyped pointer to arbitrary data your application previously passed to the function `CMIterateColorSyncFolder` (page 780).

Return Value
An `OSErr` value. If `MyCMProfileIterateCallback` returns an error, `CMIterateColorSyncFolder` stops iterating and returns the error value to its caller (presumably your code).

Discussion
When your application needs information about the profiles currently available in the profiles folder, it calls the function `CMIterateColorSyncFolder` (page 780) , which, depending on certain conditions, calls your callback routine once for each profile. See the description of `CMIterateColorSyncFolder` for information on when it calls the `MyCMProfileIterateCallback` function.

Your `MyCMProfileIterateCallback` function examines the structure pointed to by the `iterateData` parameter to obtain information about the profile it describes. The function determines whether to do anything with that profile, such as list its name in a pop-up menu of available profiles.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMApplication.h

CMProofImageProcPtr

Defines a pointer to a function that proofs an image.

```
typedef CMError (*CMProofImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf,
    CMProfileRef prfProf
);
```

If you name your function `MyCMProofImageProc`, you would declare it like this:

```
CMError MyCMProofImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 qual,
    CMProfileRef srcProf,
    UInt32 srcIntent,
    CMProfileRef dstProf,
    CMProfileRef prfProf
);
```

Parameters

specFrom
See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

repl

qual

srcProf

srcIntent

dstProf

prfProf

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMSetIndImageProfileProcPtr

Defines a pointer to a function that sets a specific embeded profile for a given image.

```
typedef CMError (*CMSetIndImageProfileProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 index,
    CMProfileRef prof
);
```

If you name your function `MyCMSetIndImageProfileProc`, you would declare it like this:

```
CMError MyCMSetIndImageProfileProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl,
    UInt32 index,
    CMProfileRef prof
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl

index

prof

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

CMUnembedImageProcPtr

Defines a pointer to a function that umembeds an ICC profile from an image.

```
typedef CMError (*CMUnembedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl
);
```

If you name your function `MyCMUnembedImageProc`, you would declare it like this:

```
CMError MyCMUnembedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    Boolean repl
```

```
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

repl

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CMValidImageProcPtr

Defines a pointer to a function that validates a specified image file.

```
typedef CMError (*CMValidImageProcPtr)
(
    const FSSpec * spec
);
```

If you name your function `MyCMValidImageProc`, you would declare it like this:

```
CMError MyCMValidImageProc (
    const FSSpec * spec
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

CountImageProfilesProcPtr

Defines a pointer to a function that counts the number of embeded profiles for a given image.

```
typedef CMError (*CountImageProfilesProcPtr)
(
    const FSSpec * spec,
    UInt32 * count
);
```

If you name your function `MyCountImageProfilesProc`, you would declare it like this:

```
CMError MyCountImageProfilesProc (
    const FSSpec * spec,
    UInt32 * count
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

count

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

EmbedImageProcPtr

Defines a pointer to an embed-image function.

```
typedef CMError (*EmbedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    CMProfileRef embedProf,
    UInt32 embedFlags
);
```

If you name your function `MyEmbedImageProc`, you would declare it like this:

```
CMError MyEmbedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    CMProfileRef embedProf,
    UInt32 embedFlags
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

embedProf
embedFlags

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

GetImageSpaceProcPtr

Defines a pointer to a get-image-space function.

```
typedef CMError (*GetImageSpaceProcPtr)
(
    const FSSpec * spec,
    OSType * space
);
```

If you name your function `MyGetImageSpaceProc`, you would declare it like this:

```
CMError MyGetImageSpaceProc (
    const FSSpec * spec,
    OSType * space
);
```

Parameters

spec
See the File Manager documentation for a description of the `FSSpec` data type.

space

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

GetIndImageProfileProcPtr

Defines a pointer to a function that obtains a color profile for an individual image..

```
typedef CMError (*GetIndImageProfileProcPtr)
(
    const FSSpec * spec,
    UInt32 index,
    CMProfileRef * prof
);
```

If you name your function `MyGetIndImageProfileProc`, you would declare it like this:

```
CMError MyGetIndImageProfileProc (
    const FSSpec * spec,
    UInt32 index,
    CMProfileRef * prof
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

index

prof

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

MatchImageProcPtr

Defines a pointer to a match-image function.

```
typedef CMError (*MatchImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 qual,
    UInt32 srcIntent,
    CMProfileRef srcProf,
    CMProfileRef dstProf,
    CMProfileRef prfProf,
    UInt32 matchFlags
);
```

If you name your function `MyMatchImageProc`, you would declare it like this:

```
CMError MyMatchImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 qual,
    UInt32 srcIntent,
    CMProfileRef srcProf,
    CMProfileRef dstProf,
```

```
    CMProfileRef prfProf,
    UInt32 matchFlags
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto

See the File Manager documentation for a description of the `FSSpec` data type.

qual

srcIntent

srcProf

dstProf

prfProf

matchFlags

Return Value

A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In

`CMScriptingPlugin.h`

SetIndImageProfileProcPtr

Defines a pointer to a function that sets a color profile for an individual image.

```
typedef CMError (*SetIndImageProfileProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 index,
    CMProfileRef prof,
    UInt32 embedFlags
);
```

If you name your function `MySetIndImageProfileProc`, you would declare it like this:

```
CMError MySetIndImageProfileProc (
    const FSSpec * specFrom,
    const FSSpec * specInto,
    UInt32 index,
    CMProfileRef prof,
    UInt32 embedFlags
);
```

Parameters

specFrom

See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

index

prof

embedFlags

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

UnembedImageProcPtr

Defines a pointer to an umembed-image function.

```
typedef CMError (*UnembedImageProcPtr)
(
    const FSSpec * specFrom,
    const FSSpec * specInto
);
```

If you name your function `MyUnembedImageProc`, you would declare it like this:

```
CMError MyUnembedImageProc (
    const FSSpec * specFrom,
    const FSSpec * specInto
);
```

Parameters

specFrom
See the File Manager documentation for a description of the `FSSpec` data type.

specInto
See the File Manager documentation for a description of the `FSSpec` data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

ValidateImageProcPtr

Defines a pointer to a validate-image function.

```
typedef CMError (*ValidateImageProcPtr)
(
    const FSSpec * spec
);
```

If you name your function `MyValidateImageProc`, you would declare it like this:

```
CMError MyValidateImageProc (
    const FSSpec * spec
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

ValidateSpaceProcPtr

Defines a pointer to a validate-space function.

```
typedef CMError (*ValidateSpaceProcPtr)
(
    const FSSpec * spec,
    OSType * space
);
```

If you name your function `MyValidateSpaceProc`, you would declare it like this:

```
CMError MyValidateSpaceProc (
    const FSSpec * spec,
    OSType * space
);
```

Parameters

spec

See the File Manager documentation for a description of the `FSSpec` data type.

space

Return Value
A `CMError` value. See "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`CMScriptingPlugin.h`

Data Types

CalibratorInfo

Contains data used to calibrate a display.

```
struct CalibratorInfo {
    UInt32 dataSize;
    CMDisplayIDType displayID;
    UInt32 profileLocationSize;
    CMProfileLocation * profileLocationPtr;
    CalibrateEventUPP eventProc;
    Boolean isGood;
};
typedef struct CalibratorInfo CalibratorInfo;
```

Fields

`dataSize`

`displayID`

`profileLocationSize`

`profileLocationPtr`

`eventProc`

`isGood`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMCalibrator.h`

CM2Header

Contains information that supports the header format specified by the ICC format specification for version 2.x profiles.

```
struct CM2Header {
    UInt32 size;
    OSType CMMType;
    UInt32 profileVersion;
    OSType profileClass;
    OSType dataColorSpace;
    OSType profileConnectionSpace;
    CMDateTime dateTime;
    OSType CS2profileSignature;
    OSType platform;
    UInt32 flags;
    OSType deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 renderingIntent;
    CMFixedXYZColor white;
    OSType creator;
    char reserved[44];
};
typedef struct CM2Header CM2Header;
```

Fields

`size`

The total size in bytes of the profile.

`CMMType`

The signature of the preferred CMM for color-matching and color-checking sessions for this profile. To avoid conflicts with other CMMs, this signature must be registered with the ICC. For the signature of the default CMM, see "Default CMM Signature" (page 977).

`profileVersion`

The version of the profile format. The first 8 bits indicate the major version number, followed by 8 bits indicating the minor version number. The following 2 bytes are reserved.

The profile version number is not tied to the version of the ColorSync Manager. Profile formats and their versions are defined by the ICC. For example, a major version change may indicate the addition of new required tags to the profile format; a minor version change may indicate the addition of new optional tags.

`profileClass`

One of the seven profile classes supported by the ICC: input, display, output, named color space, device link, color space conversion, or abstract. For the signatures representing profile classes, see "Profile Classes" (page 999).

`dataColorSpace`

The color space of the profile. Color values used to express colors of images using this profile are specified in this color space. For a list of the color space signatures, see "Color Space Signatures" (page 969).

`profileConnectionSpace`

The profile connection space, or PCS. The signatures for the two profile connection spaces supported by ColorSync, `cmXYZData` and `cmLabData`, are described in "Color Space Signatures" (page 969).

`dateTime`

The date and time when the profile was created. You can use this value to keep track of your own versions of this profile. For information on the date and time format, see `CMDateTime` (page 889).

`CS2profileSignature`

The `'acsp'` constant as required by the ICC format.

`platform`
: The signature of the primary platform on which this profile runs. For Apple Computer, this is `'APPL'`. For other platforms, refer to the International Color Consortium Profile Format Specification.

`flags`
: Flags that provide hints, such as preferred quality and speed options, to the preferred CMM. The `flags` field consists of an unsigned long data type. The 16 bits in the low word, 0-15, are reserved for use by the ICC. The 16 bits in the high word, 16-31, are available for use by color management systems. For information on how these bits are defined and how your application can set and test them, see "Flag Mask Definitions for Version 2.x Profiles" (page 983).

`deviceManufacturer`
: The signature of the manufacturer of the device to which this profile applies. This value is registered with the ICC.

`deviceModel`
: The model of this device, as registered with the ICC.

`deviceAttributes`
: Attributes that are unique to this particular device setup, such as media, paper, and ink types. The data type for this field is an array of two unsigned longs. The low word of `deviceAttributes[0]` is reserved by the ICC. The high word of `deviceAttributes[0]` and the entire word of `deviceAttributes[1]` are available for vendor use. For information on how the bits in `deviceAttributes` are defined and how your application can set and test them, see "Device Attribute Values for Version 2.x Profiles" (page 978).

`renderingIntent`
: The preferred rendering intent for the object or file tagged with this profile. Four types of rendering intent are defined: perceptual, relative colorimetric, saturation, and absolute colorimetric. The `renderingIntent` field consists of an unsigned long data type. The low word is reserved by the ICC and is used to set the rendering intent. The high word is available for use. For information on how the bits in `renderingIntent` are defined and how your application can set and test them, see "Rendering Intent Values for Version 2.x Profiles" (page 1012).

`white`
: The profile illuminant white reference point, expressed in the XYZ color space.

`creator`
: Signature identifying the profile creator.

`reserved`
: This field is reserved for future use.

Discussion

The ColorSync Manager defines the `CM2header` profile structure to support the header format specified by the ICC format specification for version 2.x profiles. For a description of `CMHeader`, the ColorSync 1.0 profile header, see `CMHeader` (page 898). To obtain a copy of the International Color Consortium Profile Format Specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/.

Your application cannot obtain a discrete profile header value using the element tag scheme available for use with elements outside the header. Instead, to set or modify values of a profile header, your application must obtain the entire profile header using the function `CMGetProfileHeader` (page 769) and replace the header using the function `CMSetProfileHeader` (page 813).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CM2Profile

```
struct CM2Profile {
    CM2Header header;
    CMTagElemTable tagTable;
    char elemData[1];
};
typedef struct CM2Profile CM2Profile;
typedef CM2Profile * CM2ProfilePtr;
```

Fields

`header`

`tagTable`

`elemData`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CM4Header

```
struct CM4Header {
    UInt32 size;
    OSType CMMType;
    UInt32 profileVersion;
    OSType profileClass;
    OSType dataColorSpace;
    OSType profileConnectionSpace;
    CMDateTime dateTime;
    OSType CS2profileSignature;
    OSType platform;
    UInt32 flags;
    OSType deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 renderingIntent;
    CMFixedXYZColor white;
    OSType creator;
    CMProfileMD5 digest;
    char reserved[28];
};
typedef struct CM4Header CM4Header;
```

Fields

`size`
`CMMType`
`profileVersion`
`profileClass`
`dataColorSpace`
`profileConnectionSpace`
`dateTime`
`CS2profileSignature`
`platform`
`flags`
`deviceManufacturer`
`deviceModel`
`deviceAttributes`
`renderingIntent`
`white`
`creator`
`digest`
`reserved`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMICCProfile.h`

CMAccelerationCalcData

```
struct CMAccelerationCalcData {
    SInt32 pixelCount;
    Ptr inputData;
    Ptr outputData;
    UInt32 reserved1;
    UInt32 reserved2;
};
typedef struct CMAccelerationCalcData CMAccelerationCalcData;
```

Fields

CMAccelerationCalcDataPtr

```
typedef CMAccelerationCalcData* CMAccelerationCalcDataPtr;
```

CMAccelerationCalcDataHdl

```
typedef CMAccelerationCalcDataPtr* CMAccelerationCalcDataHdl;
```

CMAccelerationTableData

```
struct CMAccelerationTableData {
    SInt32 inputLutEntryCount;
    SInt32 inputLutWordSize;
    Handle inputLut;
    SInt32 outputLutEntryCount;
    SInt32 outputLutWordSize;
    Handle outputLut;
    SInt32 colorLutInDim;
    SInt32 colorLutOutDim;
    SInt32 colorLutGridPoints;
    SInt32 colorLutWordSize;
    Handle colorLut;
    CMBitmapColorSpace inputColorSpace;
    CMBitmapColorSpace outputColorSpace;
    void *userData;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
    UInt32 reserved4;
    UInt32 reserved5;
};
typedef struct CMAccelerationTableData CMAccelerationTableData;
```

Fields

CMAccelerationTableDataPtr

```
typedef CMAccelerationTableData* CMAccelerationTableDataPtr;
```

CMAccelerationTableDataHdl

```
typedef CMAccelerationTableDataPtr* CMAccelerationTableDataHdl;
```

CMAdaptationMatrixType

```
struct CMAdaptationMatrixType {
    OSType typeDescriptor;
    unsigned long reserved;
    Fixed adaptationMatrix[9];
};
typedef struct CMAdaptationMatrixType CMAdaptationMatrixType;
```

Fields

`typeDescriptor`

`reserved`

`adaptationMatrix`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMAppleProfileHeader

Defines a data structure to provide access to both version 2.x and version 1.0 profiles, as specified by the International Color Consortium.

```
union CMAppleProfileHeader {
    CMHeader cm1;
    CM2Header cm2;
    CM4Header cm4;
};
typedef union CMAppleProfileHeader CMAppleProfileHeader;
```

Fields

`cm1`

A version 1.0 profile header. For a description of the ColorSync version 1.0 profile header, see `CMHeader` (page 898).

`cm2`

A current profile header. For a description of the ColorSync profile header, see `CM2Header` (page 875).

`cm4`

Discussion
The ColorSync Manager defines the `CMAppleProfileHeader` structure to provide access to both version 2.x and version 1.0 profiles, as specified by the International Color Consortium. To obtain a copy of the International Color Consortium Profile Format Specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMBitmap

Contains information that describes color bitmap images.

```
struct CMBitmap {
    char * image;
    long width;
    long height;
    long rowBytes;
    long pixelSize;
    CMBitmapColorSpace space;
    long user1;
    long user2;
};
typedef struct CMBitmap CMBitmap;
```

Fields

`image`
A pointer to a bit image.

`width`
The width of the bit image, that is, the number of pixels in a row.

`height`
The height of the bit image, that is, the number of rows in the image.

`rowBytes`
The offset in bytes from one row of the image to the next.

`pixelSize`
The number of bits per pixel. The pixel size should correspond to the packing size specified in the `space` field. This requirement is not enforced as of ColorSync version 2.5, but it may be enforced in future versions.

`space`
The color space in which the colors of the bitmap image are specified. For a description of the possible color spaces for color bitmaps, see "Color Space Constants With Packing Formats" (page 962).

`user1`
Not used by ColorSync. It is recommended that you set this field to 0.

`user2`
Not used by ColorSync. It is recommended that you set this field to 0.

Discussion
The ColorSync Manager defines a bitmap structure of type `CMBitmap` to describe color bitmap images. When your application calls the function `CWMatchColors` (page 828) , you pass a pointer to a source bitmap of type `CMBitmap` containing the image whose colors are to be matched to the color gamut of the device specified by the destination profile of the given color world. If you do not want the image color matched in place, you can also pass a pointer to a resulting bitmap of type `CMBitmap` to define and hold the color-matched image.

For QuickDraw GX, an image can have an indexed bitmap to a list of colors. The ColorSync Manager does not support indexed bitmaps in the same way QuickDraw GX does. ColorSync supports indexed bitmaps only when the `cmNamedIndexed32Space` color space constant is used in conjunction with a named color space profile.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMBitmapCallBackProc

```
typedef CMBitmapCallBackProcPtr CMBitmapCallBackProc;
```

CMBitmapCallBackUPP

Defines a universal procedure pointer to a bitmap callback.

```
typedef CMBitmapCallBackProcPtr CMBitmapCallBackUPP;
```

Discussion
For more information, see the description of the `CMBitmapCallBackProcPtr` (page 852) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMBufferLocation

```
struct CMBufferLocation {
    void * buffer;
    UInt32 size;
};
typedef struct CMBufferLocation CMBufferLocation;
```

Fields
`buffer`
`size`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMCMYColor

Contains color values expressed in the CMY color space.

```
struct CMCMYColor {
    UInt16 cyan;
    UInt16 magenta;
    UInt16 yellow;
};
typedef struct CMCMYColor CMCMYColor;
```

Fields

`cyan`

`magenta`

`yellow`

Discussion

A color value expressed in the CMY color space is composed of `cyan`, `magenta`, and `yellow` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMCMYKColor

Contains color values expressed in the CMYK color space.

```
struct CMCMYKColor {
    UInt16 cyan;
    UInt16 magenta;
    UInt16 yellow;
    UInt16 black;
};
typedef struct CMCMYKColor CMCMYKColor;
```

Fields

`cyan`

`magenta`

`yellow`

`black`

Discussion

A color value expressed in the CMYK color space is composed of `cyan`, `magenta`, `yellow`, and `black` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMColor

Defines a union that can be used to specify a color value defined by one of the 15 data types supported by the union.

```
union CMColor {
    CMRGBColor rgb;
    CMHSVColor hsv;
    CMHLSColor hls;
    CMXYZColor XYZ;
    CMLabColor Lab;
    CMLuvColor Luv;
    CMYxyColor Yxy;
    CMCMYKColor cmyk;
    CMCMYColor cmy;
    CMGrayColor gray;
    CMMultichannel5Color mc5;
    CMMultichannel6Color mc6;
    CMMultichannel7Color mc7;
    CMMultichannel8Color mc8;
    CMNamedColor namedColor;
};
typedef union CMColor CMColor;
```

Fields

`rgb`
A color value expressed in the RGB color space as data of type `CMRGBColor` (page 930).

`hsv`
A color value expressed in the HSV color space as data of type `CMHSVColor` (page 901).

`hls`
A color value expressed in the HLS color space as data of type `CMHLSColor` (page 901).

`XYZ`
A color value expressed in the XYZ color space as data of type `CMXYZColor` (page 943).

`Lab`
A color value expressed in the L*a*b* color space as data of type `CMLabColor` (page 903).

`Luv`
A color value expressed in the L*u*v* color space as data of type `CMLuvColor` (page 905).

`Yxy`
A color value expressed in the Yxy color space as data of type `CMYxyColor` (page 944).

`cmyk`
A color value expressed in the CMYK color space as data of type `CMCMYKColor` (page 884).

`cmy`
A color value expressed in the CMY color space as data of type `CMCMYColor` (page 883).

`gray`
A color value expressed in the Gray color space as data of type `CMGrayColor` (page 897).

`mc5`
A color value expressed in the five-channel multichannel color space as data of type `CMMultichannel5Color`. See `CMMultichannel5Color` (page 910) for a description of the `CMMultichannel5Color` data type.

`mc6`
A color value expressed in the six-channel multichannel color space as data of type `CMMultichannel6Color`. See `CMMultichannel6Color` (page 911) for a description of the `CMMultichannel6Color` data type.

`mc7`
: A color value expressed in the seven-channel multichannel color space as data of type `CMMultichannel7Color`. See `CMMultichannel7Color` (page 911) for a description of the `CMMultichannel7Color` data type.

`mc8`
: A color value expressed in the eight-channel multichannel color space as data of type `CMMultichannel8Color`. See `CMMultichannel8Color` (page 911) for a description of the `CMMultichannel8Color` data type.

`namedColor`
: A color value expressed as an index into a named color space. See `CMNamedColor` (page 914) for a description of the `CMNamedColor` data type.

Discussion
A color union can contain one of the above fields.

Your application can use a union of type `CMColor` to specify a color value defined by one of the 15 data types supported by the union. Your application uses an array of color unions to specify a list of colors to match, check, or convert. The array is passed as a parameter to the general purpose color matching, color checking, or color conversion functions. The following functions use a color union:

- The function `CWMatchColors` (page 828) matches the colors in the color list array to the data color space of the destination profile specified by the color world.
- The function `CWCheckColors` (page 821) checks the colors in the color list array against the color gamut specified by the color world's destination profile.
- The color conversion functions, described in "Converting Between Color Spaces", take source and destination array parameters of type `CMColor` specifying lists of colors to convert from one color space to another.

You do not use a union of type `CMColor` to convert colors expressed in the XYZ color space as values of type `CMFixedXYZ` because the `CMColor` union does not support the `CMFixedXYZ` data type.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`CMApplication.h`

CMConcatCallBackUPP

Defines a universal procedure pointer to a progress-monitoring function that the ColorSync Manager calls during lengthy color world processing.

```
typedef CMConcatCallBackProcPtr CMConcatCallBackUPP;
```

Discussion
For more information, see the description of the `CMConcatCallBackProcPtr` (page 853) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMConcatProfileSet

Contains profile and other information needed to set up a color world.

```
struct CMConcatProfileSet {
    UInt16 keyIndex;
    UInt16 count;
    CMProfileRef profileSet[1];
};
typedef struct CMConcatProfileSet CMConcatProfileSet;
```

Fields

`keyIndex`

A zero-based index into the array of profile references identifying the profile whose CMM is used for the entire session. The profile's `CMMType` field identifies the CMM.

`count`

The one-based count of profiles in the profile array. A minimum of one profile is required.

`profileSet`

A variable-length array of profile references. The references must be in processing order from source to destination. The rules governing the types of profiles you can specify in a profile array differ depending on whether you are creating a profile set for the function `CWConcatColorWorld` (page 823) or for the function `CWNewLinkProfile` (page 830). See the function descriptions for details.

Discussion

You can call the function `NCWNewColorWorld` (page 846) to create a color world for operations such as color matching and color conversion. A color world is normally based on two profiles—source and destination. But it can include a series of profiles that describe the processing for a work-flow sequence, such as scanning, printing, and previewing an image. To create a color world that includes a series of profiles, you use the function `CWConcatColorWorld` (page 823).

The array specified in the `profileSet` field identifies a concatenated profile set your application can use to establish a color world in which the sequential relationship among the profiles exists until your application disposes of the color world. Alternatively, you can create a device link profile composed of a series of linked profiles that remains intact and available for use again after your application disposes of the concatenated color world. In either case, you use a data structure of type `CMConcatProfileSet` to define the profile set.

A device link profile accommodates users who use a specific configuration requiring a combination of device profiles and possibly non-device profiles repeatedly over time.

To set up a color world that includes a concatenated set of profiles, your application uses the function `CWConcatColorWorld` (page 823) , passing it a structure of type `CMConcatProfileSet` . The array you pass may contain a set of profile references or it may contain only the profile reference of a device link profile. To create a device link profile, your application calls the function `CWNewLinkProfile` (page 830), passing a structure of type `CMConcatProfileSet`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMCurveType

```
struct CMCurveType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 countValue;
    UInt16 data[1];
};
typedef struct CMCurveType CMCurveType;
```

Fields

`typeDescriptor`

`reserved`

`countValue`

`data`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMCWInfoRecord

Contains information about a given color world.

```
struct CMCWInfoRecord {
    UInt32 cmmCount;
    CMMInfoRecord cmmInfo[2];
};
typedef struct CMCWInfoRecord CMCWInfoRecord;
```

Fields

`cmmCount`

The number of CMMs involved in the color-matching session, either 1 or 2.

`cmmInfo`

An array containing two elements. Depending on the value that `cmmCount` returns, the `cmmInfo` array contains one or two records of type `CMMInfoRecord` (page 909) reporting the CMM type and version number.

If `cmmCount` is 1, the first element of the array (`cmmInfo[0]`) describes the CMM and the contents of the second element of the array (`cmmInfo[1]`) is undefined.

If `cmmCount` is 2, the first element of the array (`cmmInfo[0]`) describes the source CMM and the second element of the array (`cmmInfo[1]`) describes the destination CMM.

Discussion
Your application supplies a color world information record structure of type `CMCWInfoRecord` as a parameter to the `CMGetCWInfo` function to obtain information about a given color world. The ColorSync Manager uses this data structure to return information about the color world.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
CMApplication.h

CMDataType

```
struct CMDataType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 dataFlag;
    char data[1];
};
typedef struct CMDataType CMDataType;
```

Fields
typeDescriptor
reserved
dataFlag
data

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMDateTime

Contains data that specifies a date and time in year, month, day of the month, hours, minutes, and seconds

```
struct CMDateTime {
    UInt16 year;
    UInt16 month;
    UInt16 dayOfTheMonth;
    UInt16 hours;
    UInt16 minutes;
    UInt16 seconds;
};
typedef struct CMDateTime CMDateTime;
```

Fields
year
: The year. Note that to indicate the year 1984, this field would store the integer 1984, not just 84.

month
: The month of the year, where 1 represents January, and 12 represents December.

dayOfTheMonth
: The day of the month, ranging from 1 to 31.

hours
: The hour of the day, ranging from 0 to 23, where 0 represents midnight and 23 represents 11:00 P.M.

minutes
: The minutes of the hour, ranging from 0 to 59.

`seconds`
The seconds of the minute, ranging from 0 to 59.

Discussion
The ColorSync Manager defines the `CMDateTime` data structure to specify a date and time in year, month, day of the month, hours, minutes, and seconds. Other ColorSync structures use the `CMDateTime` structure to specify information such as the creation date or calibration date for a color space profile.

The `CMDateTime` structure is similar to the Macintosh Toolbox structure `DateTimeRec`, and like it, is intended to hold date and time values only for a Gregorian calendar.

The `CMDateTime` structure is platform independent. However, when used with Macintosh Toolbox routines such as `SecondsToDate` and `DateToSeconds`, which use seconds to designate years, the range of years that can be represented is limited.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMDateTimeType

```
struct CMDateTimeType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMDateTime dateTime;
};
typedef struct CMDateTimeType CMDateTimeType;
```

Fields
`typeDescriptor`
`reserved`
`dateTime`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMDeviceData

```
struct CMDeviceData {
    UInt32 dataVersion;
    CMDeviceSpec deviceSpec;
    CMDeviceScope deviceScope;
    CMDeviceState deviceState;
    CMDeviceProfileID defaultProfileID;
    UInt32 profileCount;
    UInt32 reserved;
};
typedef struct CMDeviceData CMDeviceData;
```

CMDeviceDataPtr

```
typedef CMDeviceData* CMDeviceDataPtr;
```

CMDeviceID

Defines a data type for a CM device ID.

```
typedef UInt32 CMDeviceID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMDeviceIntegration.h`

CMDeviceInfo

```
struct CMDeviceInfo {
    UInt32 dataVersion;
    CMDeviceClass deviceClass;
    CMDeviceID deviceID;
    CMDeviceScope deviceScope;
    CMDeviceState deviceState;
    CMDeviceProfileID defaultProfileID;
    CFDictionaryRef * deviceName;
    UInt32 profileCount;
    UInt32 reserved;
};
typedef struct CMDeviceInfo CMDeviceInfo;
typedef CMDeviceInfo * CMDeviceInfoPtr;
```

Fields

`dataVersion`

`deviceClass`

`deviceID`

`deviceScope`

`deviceState`

`defaultProfileID`

`deviceName`

See the CFDictionary documentation for a description of the `CFDictionaryRef` data type.

`profileCount`

`reserved`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMDeviceIntegration.h`

CMDeviceName

```
struct CMDeviceName {
    UniCharCount deviceNameLength;
    UniChar deviceName[256];
};
typedef struct CMDeviceName CMDeviceName;
```

Fields

CMDeviceNamePtr

```
typedef CMDeviceName* CMDeviceNamePtr;
```

CMDeviceProfileArray

```
struct CMDeviceProfileArray {
    UInt32 profileCount;
    CMDeviceProfileInfo profiles[1];
};
typedef struct CMDeviceProfileArray CMDeviceProfileArray;
typedef CMDeviceProfileArray * CMDeviceProfileArrayPtr;
```

Fields

`profileCount`

`profiles`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMDeviceIntegration.h`

CMDeviceProfileID

```
typedef UInt32 CMDeviceProfileID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMDeviceIntegration.h`

CMDeviceProfileInfo

```
struct CMDeviceProfileInfo {
    UInt32 dataVersion;
    CMDeviceProfileID profileID;
    CMProfileLocation profileLoc;
    CFDictionaryRef profileName;
    UInt32 reserved;
};
typedef struct CMDeviceProfileInfo CMDeviceProfileInfo;
```

Fields

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMDeviceIntegration.h`

CMDeviceProfileScope

```
typedef CMDeviceScope CMDeviceProfileScope;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CMDeviceIntegration.h`

CMDeviceScope

```
struct CMDeviceScope {
    CFStringRef deviceUser;
    CFStringRef deviceHost;
};
typedef struct CMDeviceScope CMDeviceScope;
typedef CMDeviceScope CMDeviceProfileScope;
```

Fields
`deviceUser`
`deviceHost`

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`CMDeviceIntegration.h`

CMDeviceSpec

```
struct CMDeviceSpec {
    UInt32 specVersion;
    CMDeviceClass deviceClass;
    CMDeviceID deviceID;
    CMDeviceName deviceName;
    UInt32 reserved;
};
typedef struct CMDeviceSpec CMDeviceSpec;
```

Fields

CMDeviceSpecPtr

```
typedef CMDeviceSpec* CMDeviceSpecPtr;
```

CMDeviceState

```
typedef UInt32 CMDeviceState;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMDeviceIntegration.h

CMDisplayIDType

Defines a data type for a display ID type.

```
typedef UInt32 CMDisplayIDType;
```

Discussion
This data type is passed as a parameter to the functions `CMGetProfileByAVID` (page 767) and `CMSetProfileByAVID` (page 808).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMTypes.h

CMError

Defines a data type for a ColorSync Manager result code.

```
typedef  CMError;
```

Discussion
For a list of possible resutlt codes, see "ColorSync Manager Result Codes" (page 1020).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMFileLocation

Contains a file specification for a profile stored in a disk file.

```
struct CMFileLocation {
    FSSpec spec;
};
typedef struct CMFileLocation CMFileLocation;
```

Fields

`spec`

A file system specification structure giving the location of the profile file. A file specification structure includes the volume reference number, the directory ID of the parent directory, and the filename or directory name. See the File Manager documentation for a description of the `FSSpec` data type.

Discussion
Your application uses the `CMFileLocation` structure to provide a file specification for a profile stored in a disk file. You provide a file specification structure in the `CMProfileLocation` structure's `u` field to specify the location of an existing profile or a profile to be created.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMFixedXYColor

```
struct CMFixedXYColor {
    Fixed x;
    Fixed y;
};
typedef struct CMFixedXYColor CMFixedXYColor;
```

Fields

`x`

`y`

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CMICCProfile.h`

CMFixedXYZColor

Contains data that specifies the profile illuminant in the profile header's `white` field and other profile element values.

```
struct CMFixedXYZColor {
    Fixed X;
    Fixed Y;
    Fixed Z;
};
typedef struct CMFixedXYZColor CMFixedXYZColor;
```

Fields

`X`

`Y`

`Z`

Discussion

ColorSync uses the `CMFixedXYZColor` data type to specify the profile illuminant in the profile header's `white` field and to specify other profile element values. Color component values defined by the `Fixed` type definition can be used to specify a color value in the XYZ color space with greater precision than a color whose components are expressed as `CMXYZComponent` data types. The `Fixed` data type is a signed 32-bit value. A color value expressed in the XYZ color space whose color components are of type `Fixed` is defined by the `CMFixedXYZColor` type definition.

Your application can convert colors defined in the XYZ color space between `CMXYZColor` data types (in which the color components are expressed as 16-bit unsigned values) and `CMFixedXYZColor` data types (in which the colors are expressed as 32-bit signed values). To convert color values, you use the functions `CMConvertFixedXYZToXYZ` (page 730) and `CMConvertXYZToFixedXYZ` (page 736).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMFlattenUPP

Defines a universal procedure pointer to a data-flattening callback.

```
typedef CMFlattenProcPtr CMFlattenUPP;
```

Discussion

For more information, see the description of the `CMFlattenProcPtr` (page 855) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMTypes.h`

CMGrayColor

Contains a color value expressed in the gray color space.

```
struct CMGrayColor {
    UInt16 gray;
};
typedef struct CMGrayColor CMGrayColor;
```

Fields
`gray`

Discussion
A color value expressed in the Gray color space is composed of a single component, `gray`, represented as a numeric value within the range of 0 to 65535 inclusive.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMHandleLocation

Contains a handle specification for a profile stored in relocatable memory.

```
struct CMHandleLocation {
    Handle h;
};
typedef struct CMHandleLocation CMHandleLocation;
```

Fields
`h`
A data structure of type `Handle` containing a handle that indicates the location of a profile in memory.

Discussion
Your application uses the `CMHandleLocation` structure to provide a handle specification for a profile stored in relocatable memory. You provide the handle specification structure in the `CMProfileLocation` structure's `u` field to specify an existing profile or a profile to be created.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMHeader

Contains version 1.0 profile header data.

```
struct CMHeader {
    UInt32 size;
    OSType CMMType;
    UInt32 applProfileVersion;
    OSType dataType;
    OSType deviceType;
    OSType deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 profileNameOffset;
    UInt32 customDataOffset;
    CMMatchFlag flags;
    CMMatchOption options;
    CMXYZColor white;
    CMXYZColor black;
};
typedef struct CMHeader CMHeader;
```

Fields

`size`
The total size in bytes of the profile, including any custom data.

`CMMType`
The signature of the preferred CMM for color-matching and color-checking sessions for this profile. To avoid conflicts with other CMMs, this signature must be registered with the ICC. For the signature of the default CMM, see "Default CMM Signature" (page 977).

`applProfileVersion`
The Apple profile version. Set this field to $0100 (defined as the constant `kCMApplProfileVersion`).

`dataType`
The kind of color data.

`deviceType`
The kind of device.

`deviceManufacturer`
A name supplied by the device manufacturer.

`deviceModel`
The device model specified by the manufacturer.

`deviceAttributes`
Private information such as paper surface and ink temperature.

`profileNameOffset`
The offset to the profile name from the top of data.

`customDataOffset`
The offset to any custom data from the top of data.

`flags`

A field used by drivers; it can hold one of the following flags:

`CMNativeMatchingPreferredCMTurnOffCache`

The `CMNativeMatchingPreferred` flag is available for developers of intelligent peripherals that can off-load color matching into the peripheral. Most drivers will not use this flag. (Its default setting is 0, meaning that the profile creator does not care whether matching occurs on the host or the device.)

Use the `CMTurnOffCache` flag for CMMs that will not benefit from a cache, such as those that can look up data from a table with less overhead, or that do not want to take the memory hit a cache entails, or that do their own caching and do not want the CMM to do it. (The default is 0, meaning turn on cache.

`options`

The `options` field specifies the preferred matching for this profile; the default is `CMPerceptualMatch`; other values are `CMColorimetricMatch` or `CMSaturationMatch`. The options are set by the image creator.

`white`

The profile illuminant white reference point, expressed in the XYZ color space.

`black`

The black reference point for this profile, expressed in the XYZ color space.

Discussion

ColorSync 1.0 defined a version 1.0 profile whose structure and format are different from that of the ICC version 2.x profile. The `CMHeader` data type represents the version 1.0 profile header. For more information on profile version numbers, see "ColorSync and ICC Profile Format Version Numbers". To obtain a copy of the International Color Consortium Profile Format Specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/

Your application cannot use ColorSync Manager functions to update a version 1.0 profile or to search for version 1.0 profiles. However, your application can use other ColorSync Manager functions that operate on version 1.0 profiles. For example, your application can open a version 1.0 profile using the function `CMOpenProfile` (page 790), obtain the version 1.0 profile header using the function `CMGetProfileHeader` (page 769), and access version 1.0 profile elements using the function `CMGetProfileElement` (page 768).

To make it possible to operate on both version 1.0 profiles and version 2.x profiles, the ColorSync Manager defines the union `CMAppleProfileHeader` (page 881), which supports either profile -*header version. The `CMHeader` data type defines the version 1.0 profile header, while the `CM2Header` (page 875) data type defines the version 2.x profile header.

Version Notes

Use of the `CMHeader` type is not recommended for ColorSync versions starting with 2.0. Use `CM2Header` (page 875) instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMHLSColor

Contains a color value expressed in the HLS color space.

```
struct CMHLSColor {
    UInt16 hue;
    UInt16 lightness;
    UInt16 saturation;
};
typedef struct CMHLSColor CMHLSColor;
```

Fields

`hue`
A hue value that represents a fraction of a circle in which red is positioned at 0. .

`lightness`
A lightness value.

`saturation`
A saturation value.

Discussion

A color value expressed in the HLS color space is composed of `hue`, `lightness`, and `saturation` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMHSVColor

Contains a color value expressed in the HSV color space.

```
struct CMHSVColor {
    UInt16 hue;
    UInt16 saturation;
    UInt16 value;
};
typedef struct CMHSVColor CMHSVColor;
```

Fields

`hue`

`saturation`

`value`

Discussion

A color value expressed in the HSV color space is composed of `hue`, `saturation`, and `value` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 inclusive. The `hue` value represents a fraction of a circle in which red is positioned at 0.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMIntentCRDVMSize

Defines the rendering intent and its maximum VM size.

```
struct CMIntentCRDVMSize {
    long renderingIntent;
    UInt32 VMSize;
};
typedef struct CMIntentCRDVMSize CMIntentCRDVMSize;
```

Fields

`renderingIntent`
: The rendering intent whose CRD virtual memory size you want to obtain. The rendering intent values are described in "Rendering Intent Values for Version 2.x Profiles" (page 1012).

`VMSize`
: The virtual memory size of the CRD for the rendering intent specified for the `renderingIntent` field.

Discussion

To specify the maximum virtual memory (VM) size of the color rendering dictionary (CRD) for a specific rendering intent for a particular PostScript(TM) Level 2 printer type, a printer profile can include the optional Apple-defined `'psvm'` tag. The PostScript CRD virtual memory size tag structure's element data includes an array containing one entry for each rendering intent and its virtual memory size.

If a PostScript printer profile includes this tag, the default CMM uses the tag and returns the values specified by the tag when your application or device driver calls the function `CMGetPS2ColorRenderingVMSize` (page 775).

If a PostScript printer profile does not include this tag, the CMM uses an algorithm to determine the VM size of the CRD. This may result in a size that is greater than the actual VM size.

The `CMPS2CRDVMSizeType` data type for the tag includes an array containing one or more members of type `CMIntentCRDVMSize`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMIString

Defines a profile name.

```
struct CMIString {
    ScriptCode theScript;
    Str63 theString;
};
typedef struct CMIString CMIString;
typedef CMIString IString;
```

Fields

`theScript`
: The script code for the `theString` parameter.

`theString`
: The profile name.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMICCProfile.h`

CMLabColor

Contains a color value expressed in the L*a*b* color space.

```
struct CMLabColor {
    UInt16 L;
    UInt16 a;
    UInt16 b;
};
typedef struct CMLabColor CMLabColor;
```

Fields

`L`

A numeric value within the range of 0 to 65535, which maps to 0 to 100 inclusive. Note that this encoding is slightly different from the 0 to 65280 encoding of the L channel defined in the ICC specification for PCS L*a*b values.

`a`

A value that ranges from 0 to 65535, and maps to –128 to 127.996 inclusive.

`b`

A value that ranges from 0 to 65535, and maps to –128 to 127.996 inclusive.

Discussion
A color expressed in the L*a*b* color space is composed of `L`, `a`, and `b` component values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMLut16Type

```
struct CMLut16Type {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 inputChannels;
    UInt8 outputChannels;
    UInt8 gridPoints;
    UInt8 reserved2;
    Fixed matrix[3][3];
    UInt16 inputTableEntries;
    UInt16 outputTableEntries;
    UInt16 inputTable[1];
};
typedef struct CMLut16Type CMLut16Type;
```

Fields

`typeDescriptor`

`reserved`

`inputChannels`

`outputChannels`

`gridPoints`

`reserved2`

`matrix`

`inputTableEntries`

`outputTableEntries`

`inputTable`

`CLUT`

`outputTable`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMLut8Type

```
struct CMLut8Type {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 inputChannels;
    UInt8 outputChannels;
    UInt8 gridPoints;
    UInt8 reserved2;
    Fixed matrix[3][3];
    UInt8 inputTable[1];
};
typedef struct CMLut8Type CMLut8Type;
```

Fields

`typeDescriptor`
`reserved`
`inputChannels`
`outputChannels`
`gridPoints`
`reserved2`
`matrix`
`inputTable`
`CLUT`
`outputTable`
`aNet`
`aNode`
`aSocket`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMLuvColor

Contains a color value expressed in the L*u*v* color space.

```
struct CMLuvColor {
    UInt16 L;
    UInt16 u;
    UInt16 v;
};
typedef struct CMLuvColor CMLuvColor;
```

Fields

`L`
A numeric value within the range of 0 to 65535 that maps to 0 to 100 inclusive.

`u`
A numeric value within the range of 0 to 65535 that maps to –128 to 127.996 inclusive.

`v`
A numeric value within the range of 0 to 65535 that maps to –128 to 127.996 inclusive.

Discussion
A color value expressed in the L*u*v* color space is composed of `L`, `u`, and `v` component values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMMakeAndModel

Contains make and model information fro a device.

```
struct CMMakeAndModel {
    OSType manufacturer;
    UInt32 model;
    UInt32 serialNumber;
    UInt32 manufactureDate;
    UInt32 reserved1;
    UInt32 reserved2;
    UInt32 reserved3;
    UInt32 reserved4;
};
typedef struct CMMakeAndModel CMMakeAndModel;
```

Fields
`manufacturer`
`model`
`serialNumber`
`manufactureDate`
`reserved1`
`reserved2`
`reserved3`
`reserved4`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMMakeAndModelType

Contains make and model information along with a type descriptor.

```
struct CMMakeAndModelType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMMakeAndModel makeAndModel;
};
typedef struct CMMakeAndModelType CMMakeAndModelType;
```

Fields

`typeDescriptor`

`reserved`

`makeAndModel`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMMatchFlag

Defines a data type for match flags.

```
typedef  long CMMatchFlag;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMICCProfile.h`

CMMatchOption

Defines a data type for match options.

```
typedef long CMMatchOption;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMICCProfile.h`

CMMatchRef

Defines an abstract private data structure for the color-matching-session reference.

```
typedef struct OpaqueCMMatchRef * CMMatchRef;
```

Discussion
The ColorSync Manager defines an abstract private data structure of type `OpaqueCMMatchRef` for the color-matching-session reference. When your application calls the function `NCMBeginMatching` (page 838) to begin a QuickDraw-specific color-matching session, the ColorSync Manager returns a reference pointer to the color-matching session which you must later pass to the `CMEndMatching` function to conclude the session.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMTypes.h`

CMMeasurementType

Contains measurement type information.

```
struct CMMeasurementType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 standardObserver;
    CMFixedXYZColor backingXYZ;
    UInt32 geometry;
    UInt32 flare;
    UInt32 illuminant;
};
typedef struct CMMeasurementType CMMeasurementType;
```

Fields
`typeDescriptor`
`reserved`
`standardObserver`
`backingXYZ`
`geometry`
`flare`
`illuminant`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMMInfo

Contains information pertainting to a color management module.

```
struct CMMInfo {
    UInt32 dataSize;
    OSType CMMType;
    OSType CMMMfr;
    UInt32 CMMVersion;
    unsigned char ASCIIName[32];
    unsigned char ASCIIDesc[256];
    UniCharCount UniCodeNameCount;
    UniChar UniCodeName[32];
    UniCharCount UniCodeDescCount;
    UniChar UniCodeDesc[256];
};
typedef struct CMMInfo CMMInfo;
```

Fields

`dataSize`

`CMMType`

`CMMMfr`

`CMMVersion`

`ASCIIName`

`ASCIIDesc`

`UniCodeNameCount`

`UniCodeName`

`UniCodeDescCount`

`UniCodeDesc`

`TPLFMT_BKSZ`

`TPLFMT_NBLOCKS`

`TPLFMT_EDCLOC`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMMInfoRecord

Contains CMM type and version information.

```
struct CMMInfoRecord {
    OSType CMMType;
    long CMMVersion;
};
typedef struct CMMInfoRecord CMMInfoRecord;
```

Fields

`CMMType`

The signature of the CMM as specified in the profile header's `CMMType` field. The `CMGetCWInfo` function returns this value.

`CMMVersion`

The version of the CMM. The `CMGetCWInfo` function returns this value.

Discussion
Your application supplies an array containing two CMM information record structures of type `CMMInfoRecord` as a field of the `CMCWInfoRecord` structure. These structures allow the `CMGetCWInfo` function to return information about the one or two CMMs used in a given color world. Your application must allocate memory for the array. When your application calls the `CMGetCWInfo` function, it passes a pointer to the `CMCWInfoRecord` structure containing the array.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMMIterateUPP

Defines a universal procedure pointer to a CMM interation callback.

```
typedef CMMIterateProcPtr CMMIterateUPP;
```

Discussion
For more information, see the description of the `CMMIterateProcPtr` (page 862) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMMultichannel5Color

Contains a color value expressed in the multichannel color space with 5 channels.

```
struct CMMultichannel5Color {
    UInt8 components[5];
};
typedef struct CMMultichannel5Color CMMultichannel5Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 5 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMMultichannel6Color

Contains a color expressed in the multichannel color space with 6 channels.

```
struct CMMultichannel6Color {
    UInt8 components[6];
};
typedef struct CMMultichannel6Color CMMultichannel6Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 6 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMMultichannel7Color

Contains a color value expressed in the multichannel color space with 7 channels.

```
struct CMMultichannel7Color {
    UInt8 components[7];
};
typedef struct CMMultichannel7Color CMMultichannel7Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 7 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMMultichannel8Color

Contains a color value expressed in the multichannel color space with 8 channels

```
struct CMMultichannel8Color {
    UInt8 components[8];
};
typedef struct CMMultichannel8Color CMMultichannel8Color;
```

Fields
`components`

Discussion
A color expressed in the multichannel color space with 8 channels. The color value for each channel component is expressed as an unsigned byte of type `char`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMMultiFunctCLUTType

```
struct CMMultiFunctCLUTType {
    UInt8 gridPoints[16];
    UInt8 entrySize;
    UInt8 reserved[3];
    UInt8 data[1];
};
typedef struct CMMultiFunctCLUTType CMMultiFunctCLUTType;
```

Fields
`gridPoints`
`entrySize`
`reserved`
`data`

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CMICCProfile.h`

CMMultiFunctLutA2BType

```
typedef CMMultiFunctLutType CMMultiFunctLutA2BType;
```

Availability
Available in Mac OS X v10.1 through Mac OS X v10.4.

Declared In
`CMICCProfile.h`

CMMultiFunctLutB2AType

```
typedef CMMultiFunctLutType CMMultiFunctLutB2AType;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
CMICCProfile.h

CMMultiFunctLutType

```
struct CMMultiFunctLutType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 inputChannels;
    UInt8 outputChannels;
    UInt16 reserved2;
    UInt32 offsetBcurves;
    UInt32 offsetMatrix;
    UInt32 offsetMcurves;
    UInt32 offsetCLUT;
    UInt32 offsetAcurves;
    UInt8 data[1];
};
typedef struct CMMultiFunctLutType CMMultiFunctLutType;
typedef CMMultiFunctLutType CMMultiFunctLutA2BType;
```

Fields
typeDescriptor
reserved
inputChannels
outputChannels
reserved2
offsetBcurves
offsetMatrix
offsetMcurves
offsetCLUT
offsetAcurves
data

Availability
Available in Mac OS X v10.1 through Mac OS X v10.4.

Declared In
CMICCProfile.h

CMMultiLocalizedUniCodeEntryRec

```
struct CMMultiLocalizedUniCodeEntryRec {
    char languageCode[2];
    char regionCode[2];
    UInt32 textLength;
    UInt32 textOffset;
};
typedef struct CMMultiLocalizedUniCodeEntryRec CMMultiLocalizedUniCodeEntryRec;
```

Fields

`languageCode`

`regionCode`

`textLength`

`textOffset`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMMultiLocalizedUniCodeType

```
struct CMMultiLocalizedUniCodeType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 entryCount;
    UInt32 entrySize;
};
typedef struct CMMultiLocalizedUniCodeType CMMultiLocalizedUniCodeType;
```

Fields

`typeDescriptor`

`reserved`

`entryCount`

`entrySize`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMNamedColor

Contains a color value expressed in a named color space.

```
struct CMNamedColor {
    UInt32 namedColorIndex;
};
typedef struct CMNamedColor CMNamedColor;
```

Fields
`namedColorIndex`

Discussion
A color value expressed in a named color space is composed of a single component, `namedColorIndex`, represented as a numeric value within the range of an unsigned long, or 1 to 232 – 1 inclusive.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMNamedColor2EntryType

```
struct CMNamedColor2EntryType {
    UInt8 rootName[32];
    UInt16 PCSColorCoords[3];
    UInt16 DeviceColorCoords[1];
};
typedef struct CMNamedColor2EntryType CMNamedColor2EntryType;
```

Fields
`rootName`
`PCSColorCoords`
`DeviceColorCoords`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMNamedColor2Type

```
struct CMNamedColor2Type {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 vendorFlag;
    UInt32 count;
    UInt32 deviceChannelCount;
    UInt8 prefixName[32];
    UInt8 suffixName[32];
    char data[1];
};
typedef struct CMNamedColor2Type CMNamedColor2Type;
```

Fields

`typeDescriptor`

`reserved`

`vendorFlag`

`count`

`deviceChannelCount`

`prefixName`

`suffixName`

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMNamedColorType

```
struct CMNamedColorType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 vendorFlag;
    UInt32 count;
    UInt8 prefixName[1];
};
typedef struct CMNamedColorType CMNamedColorType;
```

Fields

`typeDescriptor`

`reserved`

`vendorFlag`

`count`

`prefixName`

`suffixName`

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMNativeDisplayInfo

Contains color information for a native display.

```
struct CMNativeDisplayInfo {
    UInt32 dataSize;
    CMFixedXYColor redPhosphor;
    CMFixedXYColor greenPhosphor;
    CMFixedXYColor bluePhosphor;
    CMFixedXYColor whitePoint;
    Fixed redGammaValue;
    Fixed greenGammaValue;
    Fixed blueGammaValue;
    UInt16 gammaChannels;
    UInt16 gammaEntryCount;
    UInt16 gammaEntrySize;
    char gammaData[1];
};
typedef struct CMNativeDisplayInfo CMNativeDisplayInfo;
```

Fields
dataSize
redPhosphor
greenPhosphor
bluePhosphor
whitePoint
redGammaValue
greenGammaValue
blueGammaValue
gammaChannels
gammaEntryCount
gammaEntrySize
gammaData

Availability
Available in Mac OS X v10.1 and later.

Declared In
CMICCProfile.h

CMNativeDisplayInfoType

Contins color information and a tpe descriptor for a native display.

```
struct CMNativeDisplayInfoType {
    OSType typeDescriptor;
    unsigned long reserved;
    CMNativeDisplayInfo nativeDisplayInfo;
};
typedef struct CMNativeDisplayInfoType CMNativeDisplayInfoType;
```

Fields

`typeDescriptor`

`reserved`

`nativeDisplayInfo`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMICCProfile.h`

CMParametricCurveType

```
struct CMParametricCurveType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt16 functionType;
    UInt16 reserved2;
    Fixed value[1];
};
typedef struct CMParametricCurveType CMParametricCurveType;
```

Fields

`typeDescriptor`

`reserved`

`functionType`

`reserved2`

`value`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMICCProfile.h`

CMPathLocation

Contains path information.

```
struct CMPathLocation {
    char path[256];
};
typedef struct CMPathLocation CMPathLocation;
```

Fields
`path`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMProcedureLocation

Contains a universal procedure pointer to a profile access procedure.

```
struct CMProcedureLocation {
    CMProfileAccessUPP proc;
    void * refCon;
};
typedef struct CMProcedureLocation CMProcedureLocation;
```

Fields
`proc`
A universal procedure pointer to a profile access procedure. For a description of the procedure, see the function `CMProfileAccessProcPtr` (page 862).

`refCon`
A pointer to the profile access procedure's private data, such as a file or resource name, a pointer to a current offset, and so on.

Discussion
Your application uses the `CMProcedureLocation` structure to provide a universal procedure pointer to a profile access procedure. You provide this structure in the `CMProfileLocation` structure's `u` field. The `CMProcedureLocation` structure also contains a pointer field to specify data associated with the profile access procedure.

The ColorSync Manager calls your profile access procedure when the profile is created, initialized, opened, read, updated, or closed.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMProfile

```
struct CMProfile {
    CMHeader header;
    CMProfileChromaticities profile;
    CMProfileResponse response;
    CMIString profileName;
    char customData[1];
};
typedef struct CMProfile CMProfile;
typedef CMProfile * CMProfilePtr;
```

Fields

`header`

`profile`

`response`

`profileName`

`customData`

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMProfileAccessUPP

Defines a universal procedure pointer to a profile access callback.

```
typedef CMProfileAccessProcPtr CMProfileAccessUPP;
```

Discussion

For more information, see the description of the `CMProfileAccessProcPtr` (page 862)) callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMTypes.h`

CMProfileChromaticities

```
struct CMProfileChromaticities {
    CMXYZColor red;
    CMXYZColor green;
    CMXYZColor blue;
    CMXYZColor cyan;
    CMXYZColor magenta;
    CMXYZColor yellow;
};
typedef struct CMProfileChromaticities CMProfileChromaticities;
```

Fields

`red`

`green`

`blue`

`cyan`

`magenta`

`yellow`

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMICCProfile.h`

CMProfileFilterProc

```
typedef CMProfileFilterProcPtr CMProfileFilterProc;
```

CMProfileFilterUPP

Defines a universal procedure pointer to a profile filter callback.

```
typedef CMProfileFilterProcPtr CMProfileFilterUPP;
```

Discussion

For more information, see the description of the `CMProfileFilterProcPtr` (page 864) callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMTypes.h`

CMProfileIdentifier

Contains data that can identify a profile but that takes up much less space than a large profile.

```
struct CMProfileIdentifier {
    CM2Header profileHeader;
    CMDateTime calibrationDate;
    UInt32 ASCIIProfileDescriptionLen;
    char ASCIIProfileDescription[1];
};
typedef struct CMProfileIdentifier CMProfileIdentifier;
typedef CMProfileIdentifier * CMProfileIdentifierPtr;
```

Fields

`profileHeader`

A version 2.x profile header structure. For more information, see `CM2Header` (page 875). In determining a profile match, all header fields are considered, except for primary platform, flags, and rendering intent.

`calibrationDate`

A structure of type `CMDateTime` (page 889), which specifies year, month, day of month, hours, minutes, and seconds. This field is optional—when set to 0, it is not considered in determining a profile match. When nonzero, it is compared to the `'calt'` tag data.

`ASCIIProfileDescriptionLen`

The length of the ASCII description string that follows.

`ASCIIProfileDescription`

The ASCII profile description string, as specified by the profile description tag.

Discussion

Embedding a profile in an image guarantees that the image can be rendered correctly on a different system. However, profiles can be large—as much as several hundred kilobytes. The ColorSync Manager defines a profile identifier structure, `CMProfileIdentifier`, that can identify a profile but that takes up much less space than a large profile.

The profile identifier structure contains a profile header, an optional calibration date, a profile description string length, and a variable-length profile description string. Your application might use an embedded profile identifier, for example, to change just the rendering intent or the flag values in an image without having to embed an entire copy of a profile. Rendering intent is described in "Rendering Intent Values for Version 2.x Profiles" (page 1012) and flag values are described in "Flag Mask Definitions for Version 2.x Profiles" (page 983).

A document containing an embedded profile identifier cannot necessarily be ported to different systems or platforms.

The ColorSync Manager provides the function routine `NCMUseProfileComment` (page 843) to embed profiles and profile identifiers in an open picture file. Your application can embed profile identifiers in place of entire profiles, or in addition to them. A profile identifier can refer to an embedded profile or to a profile on disk.

The ColorSync Manager provides two routines for finding a profile identifier:

- `CMProfileIdentifierListSearch` (page 793) for finding a profile identifier in a list of profile identifiers
- `CMProfileIdentifierFolderSearch` (page 792) for finding a profile identifier in the ColorSync Profiles folder.

The descriptions of those functions provide information on searching algorithms. See also `CMProfileSearchRef` (page 927)

The `CMProfileIdentifierPtr` type definition defines a pointer to a profile identifier structure.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMProfileIterateData

Contains a callback routine with a description of a profile that is during an iteration through the available profiles.

```
struct CMProfileIterateData {
    UInt32 dataVersion;
    CM2Header header;
    ScriptCode code;
    Str255 name;
    CMProfileLocation location;
    UniCharCount uniCodeNameCount;
    UniChar * uniCodeName;
    unsigned char * asciiName;
    CMMakeAndModel * makeAndModel;
    CMProfileMD5 * digest;
};
typedef struct CMProfileIterateData CMProfileIterateData;
```

Fields

`dataVersion`
A value identifying the version of the structure. Currently set to `cmProfileIterateDataVersion1`.

`header`
A ColorSync version 2.x profile header structure of type `CM2Header` (page 875), containing information such as the profile size, type, version, and so on.

`code`
A script code identifying the script system used for the profile description. The `ScriptCode` data type is defined in the MacTypes.h header file.

`name`
The profile name, stored as a Pascal-type string (with length byte first) of up to 255 characters.

`location`
A structure specifying the profile location. With ColorSync 2.5, the location is always file-based, but that may not be true for future versions. Your code should always verify that the location structure contains a file specification before attempting to use it.

```
uniCodeNameCount
uniCodeName
asciiName
makeAndModel
digest
TPLDEV_TYPE_WPS_SPEED
deviceData
```

Discussion
The ColorSync Manager defines the `CMProfileIterateData` structure to provide your `CMProfileIterateProcPtr` (page 865) callback routine with a description of a profile during an iteration through the available profiles that takes place when you call `CMIterateColorSyncFolder` (page 780).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMProfileIterateUPP

Defines a universal procedure pointer to a profile iteration callback.

```
typedef CMProfileIterateProcPtr CMProfileIterateUPP;
```

Discussion
For more information, see the description of the `CMProfileIterateProcPtr` (page 865)) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMProfileLocation

Contains profile location information.

```
struct CMProfileLocation {
    short locType;
    CMProfLoc u;
};
typedef struct CMProfileLocation CMProfileLocation;
```

Fields
`locType`
The type of data structure that the `u` field's `CMProfLoc` union holds—a file specification, a handle, a pointer, or a universal procedure pointer. To specify the type, you use the constants defined in the enumeration described in "Profile Location Type" (page 1003).

`u`
A union of type `CMProfLoc` (page 928) identifying the profile location.

Discussion
Your application passes a profile location structure of type `CMProfileLocation` when it calls:

- the function `CMOpenProfile` (page 790), specifying the location of a profile to open
- the `CMNewProfile` (page 788), `CWNewLinkProfile` (page 830), or `CMCopyProfile` (page 740) functions, specifying the location of a profile to create or duplicate

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMApplication.h`

CMProfileMD5

Defines a data type for an MD5 digest.

```
typedef  unsigned char CMProfileMD5[16];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMProfileName

Contains profile name and length.

```
struct CMProfileName {
    UniCharCount profileNameLength;
    UniChar profileName[256];
};
typedef struct CMProfileName CMProfileName;
```

CMProfileNamePtr

Defines a poitner to a profile name data structure.

```
typedef CMProfileName* CMProfileNamePtr;
```

CMProfileRef

Defines a reference to an opaque data type that specifies profile information.

```
typedef struct OpaqueCMProfileRef * CMProfileRef;
```

Discussion
A profile reference is the means by which your application gains access to a profile. Several ColorSync Manager functions return a profile reference to your application. Your application then passes it as a parameter on subsequent calls to other ColorSync Manager functions that use profiles.

The ColorSync Manager returns a unique profile reference in response to each individual call to the `CMOpenProfile` (page 790), `CMCopyProfile` (page 740) , and `CMNewProfile` (page 788) functions. This allows multiple applications concurrent access to a profile. The ColorSync Manager defines an abstract private data structure of type `OpaqueCMProfileRef` for the profile reference.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMProfileResponse

```
struct CMProfileResponse {
    UInt16 counts[9];
    UInt16 data[1];
};
typedef struct CMProfileResponse CMProfileResponse;
```

Fields
`counts`
`data`

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMICCProfile.h`

CMProfileSearchRecord

```
struct CMProfileSearchRecord {
    CMHeader header;
    UInt32 fieldMask;
    UInt32 reserved[2];
};
typedef struct CMProfileSearchRecord CMProfileSearchRecord;
typedef CMProfileSearchRecord * CMProfileSearchRecordPtr;
```

Fields
`header`
`fieldMask`
`reserved`

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMProfileSearchRef

Defines a reference to an opaque profile search object.

```
typedef struct OpaqueCMProfileSearchRef * CMProfileSearchRef;
```

Discussion
A search result consists of a list of profiles matching certain search criteria. When your application calls the function `CMNewProfileSearch` (page 789) to search in the ColorSync Profiles folder for profiles that meet certain criteria, the ColorSync Manager returns a reference to an internal private data structure containing the search result. Your application passes the search result reference to these ColorSync functions:

- `CMUpdateProfileSearch` (page 817) updates a search result list.
- `CMDisposeProfileSearch` (page 745) disposes of a search result list.
- `CMSearchGetIndProfile` (page 798) opens a reference to a profile at a specific position in a search result list.
- `CMSearchGetIndProfileFileSpec` (page 799) obtains the file specification for a profile in a search result list.

The ColorSync Manager uses an abstract private data structure of type `OpaqueCMProfileSearchRef` in defining the search result reference.

Version Notes
This type is not recommended for use in ColorSync 2.5.

This type does not take advantage of the profile cache added in ColorSync version 2.5. It is used with the searching described in "Searching for Profiles Prior to ColorSync 2.5". See `CMProfileIterateData` (page 923) for information on data structures used with searching in version 2.5.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMTypes.h`

CMProfileSequenceDescType

```
struct CMProfileSequenceDescType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 count;
    char data[1];
};
typedef struct CMProfileSequenceDescType CMProfileSequenceDescType;
```

Fields

`typeDescriptor`

`reserved`

`count`

`data`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMProfLoc

Defines a uniont that identifies the location of a profile.

```
union CMProfLoc {
    CMFileLocation fileLoc;
    CMHandleLocation handleLoc;
    CMPtrLocation ptrLoc;
    CMProcedureLocation procLoc;
    CMPathLocation pathLoc;
    CMBufferLocation bufferLoc;
};
typedef union CMProfLoc CMProfLoc;
```

Fields

`fileLoc`
A data structure containing a file system specification record specifying the location of a profile disk file.

`handleLoc`
A data structure containing a handle that indicates the location of a profile in relocatable memory.

`ptrLoc`
A data structure containing a pointer that points to a profile in nonrelocatable memory.

`procLoc`
A data structure containing a universal procedure pointer that points to a profile access procedure supplied by you. The ColorSync Manager calls your procedure when the profile is created, initialized, opened, read, updated, or closed.

`pathLoc`
`bufferLoc`

Discussion
You use a union of type `CMProfLoc` to identify the location of a profile. You specify the union in the `u` field of the data type `CMProfileLocation` (page 924). Your application passes a pointer to a `CMProfileLocation` structure when it calls the `CMOpenProfile` (page 790) function to identify the location of a profile or the `CMNewProfile` (page 788), `CMCopyProfile` (page 740) , or `CWNewLinkProfile` (page 830) functions to specify the location for a newly created profile.

You also pass a pointer to a `CMProfileLocation` structure to the `NCMGetProfileLocation` (page 841) and `CMGetProfileLocation` (page 770) functions to get the location of an existing profile. The `NCMGetProfileLocation` function is available starting with ColorSync version 2.5. It differs from its predecessor, `CMGetProfileLocation`, in that the newer version has a parameter for the size of the location structure for the specified profile.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`CMApplication.h`

CMPS2CRDVMSizeType

Defines the Apple-defined `'psvm'` optional tag.

```
struct CMPS2CRDVMSizeType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 count;
    CMIntentCRDVMSize intentCRD[1];
};
typedef struct CMPS2CRDVMSizeType CMPS2CRDVMSizeType;
```

Fields
`typeDescriptor`
The `'psvm'` tag signature.
`reserved`
Reserved for future use.
`count`
The number of entries in the `intentCRD` array. You should specify at least four entries: 0, 1, 2, and 3.
`intentCRD`
A variable-sized array of four or more members defined by the `CMIntentCRDSize` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMPtrLocation

Contains a pointer specification for a profile stored in nonrelocatable memory.

```
struct CMPtrLocation {
    Ptr p;
};
typedef struct CMPtrLocation CMPtrLocation;
```

Fields

`p`

A data structure of type `Ptr` holding a pointer that points to the location of a profile in memory.

Discussion

Your application uses the `CMPtrLocation` structure to provide a pointer specification for a profile stored in nonrelocatable memory. You provide the pointer specification structure in the `CMProfileLocation` structure's `u` field to point to an existing profile.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CMApplication.h`

CMRGBColor

Contains a color value expressed in the RGB color space.

```
struct CMRGBColor {
    UInt16 red;
    UInt16 green;
    UInt16 blue;
};
typedef struct CMRGBColor CMRGBColor;
```

Fields

`red`

`green`

`blue`

Discussion

A color value expressed in the RGB color space is composed of `red`, `green`, and `blue` component values. Each color component is expressed as a numeric value within the range of 0 to 65535.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

CMS15Fixed16ArrayType

```
struct CMS15Fixed16ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    Fixed value[1];
};
typedef struct CMS15Fixed16ArrayType CMS15Fixed16ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMScreeningChannelRec

```
struct CMScreeningChannelRec {
    Fixed frequency;
    Fixed angle;
    UInt32 spotFunction;
};
typedef struct CMScreeningChannelRec CMScreeningChannelRec;
```

Fields

`frequency`

`angle`

`spotFunction`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMScreeningType

```
struct CMScreeningType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 screeningFlag;
    UInt32 channelCount;
    CMScreeningChannelRec channelInfo[1];
};
typedef struct CMScreeningType CMScreeningType;
```

Fields

`typeDescriptor`

`reserved`

`screeningFlag`

`channelCount`

`data`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMSearchRecord

Contains information needed for a search.

```
struct CMSearchRecord {
    OSType CMMType;
    OSType profileClass;
    OSType dataColorSpace;
    OSType profileConnectionSpace;
    UInt32 deviceManufacturer;
    UInt32 deviceModel;
    UInt32 deviceAttributes[2];
    UInt32 profileFlags;
    UInt32 searchMask;
    CMProfileFilterUPP filter;
};
typedef struct CMSearchRecord CMSearchRecord;
```

Fields

`CMMType`

The signature of a CMM. The signature of the default CMM is specified by the `kDefaultCMMSignature` constant.

`profileClass`

The class signature identifying the type of profile to search for. For a list of profile class signatures, see "Profile Classes" (page 999).

`dataColorSpace`

A data color space. For a list of the color space signatures, see "Color Space Signatures" (page 969).

`profileConnectionSpace`

A profile connection color space. The signatures for the two profile connection spaces supported by ColorSync, `cmXYZData` and `cmLabData`, are described in "Color Space Signatures" (page 969).

`deviceManufacturer`
The signature of the manufacturer.

`deviceModel`
The model of a device.

`deviceAttributes`
Attributes for a particular device setup, such as media, paper, and ink types.

`profileFlags`
Flags that indicate hints for the preferred CMM, such as quality, speed, and memory options. In most cases, you will not want to search for profiles based on the flags settings.

`searchMask`
A bitmask that specifies the search record fields to use in the profile search.

`filter`
A pointer to an application-supplied function that determines whether to exclude a profile from the profile search result list. For more information, see the function `CMProfileFilterProcPtr` (page 864).

Discussion
Your application supplies a search record of type `CMSearchRecord` as the `searchSpec` parameter to the function `CMNewProfileSearch` (page 789). The search record structure provides the ColorSync Manager with search criteria to use in determining which version 2.x profiles to include in the result list and which to filter out.

Most of the fields in the `CMSearchRecord` structure are identical to corresponding fields in the `CM2Header` structure for version 2.x profiles. When you set a bit in the `searchMask` field of the `CMSearchRecord` structure, you cause the search criteria to include the data specified by that bit. For example, if you set the `cmMatchProfileCMMType` bit, the search result will not include a profile unless the data in the profile header's `CMMType` field matches the data you specify in the `CMSearchRecord` structure's `CMMType` field.

If you specify a bit in the `searchMask` field, you must supply information in the `CMSearchRecord` field that corresponds to that bit.

The ColorSync Manager preserves the search criteria internally along with the search result list until your application calls the `CMDisposeProfileSearch` function to release the memory. This allows your application to call the `CMUpdateProfileSearch` function to update the search result if the ColorSync Profiles folder contents change without needing to provide the search specification again.

Version Notes
This type is not recommended for use in ColorSync 2.5.

You cannot use the ColorSync Manager search functions to search for ColorSync 1.0 profiles.

This type does not take advantage of the profile cache added in ColorSync version 2.5. It is used with the searching described in "Searching for Profiles Prior to ColorSync 2.5". See `CMProfileIterateData` (page 923) for information on data structures used with searching in version 2.5.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CMApplication.h`

CMSignatureType

```
struct CMSignatureType {
    OSType typeDescriptor;
    UInt32 reserved;
    OSType signature;
};
typedef struct CMSignatureType CMSignatureType;
```

Fields

`typeDescriptor`

`reserved`

`signature`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMTagElemTable

```
struct CMTagElemTable {
    UInt32 count;
    CMTagRecord tagList[1];
};
typedef struct CMTagElemTable CMTagElemTable;
```

Fields

`count`

`tagList`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMTagRecord

```
struct CMTagRecord {
    OSType tag;
    UInt32 elementOffset;
    UInt32 elementSize;
};
typedef struct CMTagRecord CMTagRecord;
```

Fields

`tag`

`elementOffset`

`elementSize`

Availability

Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMTextDescriptionType

```
struct CMTextDescriptionType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 ASCIICount;
    UInt8 ASCIIName[2];
};
typedef struct CMTextDescriptionType CMTextDescriptionType;
```

Fields
typeDescriptor
reserved
ASCIICount
ASCIIName
UniCodeCode
UniCodeCount
UniCodeName
ScriptCodeCode
ScriptCodeCount
ScriptCodeName

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMTextType

```
struct CMTextType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 text[1];
};
typedef struct CMTextType CMTextType;
```

Fields
typeDescriptor
reserved
text

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMU16Fixed16ArrayType

```
struct CMU16Fixed16ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 value[1];
};
typedef struct CMU16Fixed16ArrayType CMU16Fixed16ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUcrBgType

```
struct CMUcrBgType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 ucrCount;
    UInt16 ucrValues[1];
};
typedef struct CMUcrBgType CMUcrBgType;
```

Fields

`typeDescriptor`

`reserved`

`ucrCount`

`ucrValues`

`bgCount`

`bgValues`

`ucrbgASCII`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt16ArrayType

```
struct CMUInt16ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt16 value[1];
};
typedef struct CMUInt16ArrayType CMUInt16ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt32ArrayType

```
struct CMUInt32ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 value[1];
};
typedef struct CMUInt32ArrayType CMUInt32ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt64ArrayType

```
struct CMUInt64ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt32 value[1];
};
typedef struct CMUInt64ArrayType CMUInt64ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUInt8ArrayType

```
struct CMUInt8ArrayType {
    OSType typeDescriptor;
    UInt32 reserved;
    UInt8 value[1];
};
typedef struct CMUInt8ArrayType CMUInt8ArrayType;
```

Fields

`typeDescriptor`

`reserved`

`value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMUnicodeTextType

```
struct CMUnicodeTextType {
    OSType typeDescriptor;
    UInt32 reserved;
    UniChar text[1];
};
typedef struct CMUnicodeTextType CMUnicodeTextType;
```

Fields

`typeDescriptor`

`reserved`

`text`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMVideoCardGamma

Contains video gamma data to store with a video gamma profile tag.

```
struct CMVideoCardGamma {
    UInt32 tagType
    union {
        CMVideoCardGammaTable table;
        CMVideoCardGammaFormula formula;
    } u;
};
typedef struct CMVideoCardGamma CMVideoCardGamma;
```

Fields

`tagType`

A "Video Card Gamma Storage Types" (page 1018) constant that specifies the format of the data currently stored in the union. To determine the type of structure present in a specific instance of the `CMVideoCardGamma` structure, you test this union tag. If you are setting up a `CMVideoCardGamma` structure to store video card gamma data, you set `tagType` to a constant value that identifies the structure type you are using. The possible constant values are described in "Video Card Gamma Storage Types" (page 1018).

`table`

A structure of type `CMVideoCardGammaTable`. If the `tagType` field has the value `cmVideoCardGammaTableType`, the `CMVideoCardGamma` structure's union field should be treated as a table, as described in `CMVideoCardGammaTable` (page 941).

`formula`

Discussion
The ColorSync Manager defines the `CMVideoCardGamma` data structure to specify the video gamma data to store with a video gamma profile tag. The structure is a union that can store data in either table or formula format.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMVideoCardGammaFormula

```
struct CMVideoCardGammaFormula {
    Fixed redGamma;
    Fixed redMin;
    Fixed redMax;
    Fixed greenGamma;
    Fixed greenMin;
    Fixed greenMax;
    Fixed blueGamma;
    Fixed blueMin;
    Fixed blueMax;
};
typedef struct CMVideoCardGammaFormula CMVideoCardGammaFormula;
```

Fields

`redGamma`
The gamma value for red. It must be greater than 0.0.

`redMin`
The minimum gamma value for red. It must be greater than 0.0 and less than 1.0.

`redMax`
The maximum gamma value for red. It must be greater than 0.0 and less than 1.0.

`greenGamma`
The gamma value for green. It must be greater than 0.0.

`greenMin`
The minimum gamma value for green. It must be greater than 0.0 and less than 1.0.

`greenMax`
The maximum gamma value for green. It must be greater than 0.0 and less than 1.0.

`blueGamma`
The gamma value for blue. It must be greater than 0.0.

`blueMin`
The minimum gamma value for blue. It must be greater than 0.0 and less than 1.0.

`blueMax`
The maximum gamma value for blue. It must be greater than 0.0 and less than 1.0.

Discussion
The ColorSync Manager defines the `CMVideoCardGammaFormula` data structure to specify video card gamma data by providing three values each for red, blue and green gamma. The values represent the actual gamma, the minimum gamma, and the maximum gamma for each color. Specifying video gamma information by formula takes less space than specifying it with a table, but the results may be less precise.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CMICCProfile.h

CMVideoCardGammaTable

```
struct CMVideoCardGammaTable {
    UInt16 channels;
    UInt16 entryCount;
    UInt16 entrySize;
    char data[1];
};
typedef struct CMVideoCardGammaTable CMVideoCardGammaTable;
```

Fields

`channels`

Number of gamma channels (1 or 3). If `channels` is set to 1 then the red, green, and blue lookup tables (LUTs) of the video card will be loaded with the same data. If `channels` is set to 3, then if the video card supports separate red, green, and blue LUTs, then the video card LUTs will be loaded with the data for the three channels from the `data` array.

`entryCount`

Number of entries per channel (1-based). The number of entries must be greater than or equal to 2.

`entrySize`

Size in bytes of each entry.

`data`

Variable-sized array of data. The size of the data is equal to `channels*entryCount*entrySize`.

Discussion

The ColorSync Manager defines the `CMVideoCardGammaTable` data structure to specify video card gamma data in table format. You specify the number of channels, the number of entries per channel, and the size of each entry. The last field in the structure is an array of size one that serves as the start of the table data. The actual size of the array is equal to the number of channels times the number of entries times the size of each entry.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMVideoCardGammaType

Specifies a video card gamma profile tag.

```
struct CMVideoCardGammaType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMVideoCardGamma gamma;
};
typedef struct CMVideoCardGammaType CMVideoCardGammaType;
```

Fields

`typeDescriptor`

The signature type for a video card gamma tag. There is currently only one type possible, `cmSigVideoCardGammaType`.

`reserved`

This field is reserved and must contain the value 0.

`gamma`

A structure that specifies the video card gamma data for the profile tag, as described in `CMVideoCardGamma` (page 939).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMViewingConditionsType

```
struct CMViewingConditionsType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMFixedXYZColor illuminant;
    CMFixedXYZColor surround;
    UInt32 stdIlluminant;
};
typedef struct CMViewingConditionsType CMViewingConditionsType;
```

Fields
`typeDescriptor`
`reserved`
`illuminant`
`surround`
`stdIlluminant`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMWorldRef

Defines an opaque data type used for color-matching and color-checking sessions.

```
typedef struct OpaqueCMWorldRef * CMWorldRef;
```

Discussion
Your application passes a color world reference as a parameter on calls to functions to perform color-matching and color-checking sessions and to dispose of the color world. When your application calls the function `NCWNewColorWorld` (page 846) and the function `CWConcatColorWorld` (page 823) to allocate a color world for color-matching and color-checking sessions, the ColorSync Manager returns a reference to the color world. The ColorSync Manager defines an abstract private data structure of type `OpaqueCMWorldRef` for the color world reference.

The color world is affected by the rendering intent, lookup flag, gamut flag, and quality flag of the profiles that make up the color world. For more information, see "Rendering Intent Values for Version 2.x Profiles" (page 1012), "Flag Mask Definitions for Version 2.x Profiles" (page 983), and "Quality Flag Values for Version 2.x Profiles" (page 1011).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMTypes.h`

CMXYZColor

Contains values for a color specified in XYZ color space.

```
struct CMXYZColor {
    CMXYZComponent X;
    CMXYZComponent Y;
    CMXYZComponent Z;
};
typedef struct CMXYZColor CMXYZColor;
typedef CMXYZColor XYZColor;
```

Fields
`X`
`Y`
`Z`

Discussion
Three color component values defined by the `CMXYZComponent` type definition combine to form a color value specified in the XYZ color space. The color value is defined by the `CMXYZColor` type definition.

Your application uses the `CMXYZColor` data structure to specify a color value in the `CMColor` union to use in general purpose color matching, color checking, or color conversion. You also use the `CMXYZColor` data structure to specify the XYZ white point reference used in the conversion of colors to or from the XYZ color space.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMXYZComponent

```
typedef UInt16 CMXYZComponent;
```

Discussion
Three components combine to express a color value defined by the `CMXYZColor` type definition in the XYZ color space. Each color component is described by a numeric value defined by the `CMXYZComponent` type definition. A component value of type `CMXYZComponent` is expressed as a 16-bit value. This is formatted as an unsigned value with 1 bit of integer portion and 15 bits of fractional portion.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CMICCProfile.h`

CMXYZType

```
struct CMXYZType {
    OSType typeDescriptor;
    UInt32 reserved;
    CMFixedXYZColor XYZ[1];
};
typedef struct CMXYZType CMXYZType;
```

Fields

`typeDescriptor`

`reserved`

`XYZ`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMICCProfile.h`

CMYKColor

```
typedef CMCMYKColor CMYKColor;
```

CMYxyColor

Contains values for a color expresed in the Yxy color space.

```
struct CMYxyColor {
    UInt16 capY;
    UInt16 x;
    UInt16 y;
};
typedef struct CMYxyColor CMYxyColor;
```

Fields

`capY`

`x`

`y`

Discussion

A color value expressed in the Yxy color space is composed of `capY`, `x`, and `y` component values. Each color component is expressed as a numeric value within the range of 0 to 65535 which maps to 0 to 1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

NCMConcatProfileSet

```
struct NCMConcatProfileSet {
    OSType cmm;
    UInt32 flags;
    UInt32 flagsMask;
    UInt32 profileCount;
    NCMConcatProfileSpec profileSpecs[1];
};
typedef struct NCMConcatProfileSet NCMConcatProfileSet;
```

Fields

`cmm`

`flags`

`flagsMask`

`profileCount`

`profileSpecs`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

NCMConcatProfileSpec

```
struct NCMConcatProfileSpec {
    UInt32 renderingIntent;
    UInt32 transformTag;
    CMProfileRef profile;
};
typedef struct NCMConcatProfileSpec NCMConcatProfileSpec;
```

Fields

`renderingIntent`

`transformTag`

`profile`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CMApplication.h`

NCMDeviceProfileInfo

```
struct NCMDeviceProfileInfo {
    UInt32 dataVersion;
    CMDeviceProfileID profileID;
    CMProfileLocation profileLoc;
    CFDictionaryRef profileName;
    CMDeviceProfileScope profileScope;
    UInt32 reserved;
};
typedef struct NCMDeviceProfileInfo NCMDeviceProfileInfo;
```

Fields

`dataVersion`

`profileID`

`profileLoc`

`profileName`

`profileScope`

`reserved`

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CMDeviceIntegration.h`

Constants

Abstract Color Space Constants

Specify values that represent general color spaces.

```
enum {
    cmNoSpace = 0x0000,
    cmRGBSpace = 0x0001,
    cmCMYKSpace = 0x0002,
    cmHSVSpace = 0x0003,
    cmHLSSpace = 0x0004,
    cmYXYSpace = 0x0005,
    cmXYZSpace = 0x0006,
    cmLUVSpace = 0x0007,
    cmLABSpace = 0x0008,
    cmReservedSpace1 = 0x0009,
    cmGraySpace = 0x000A,
    cmReservedSpace2 = 0x000B,
    cmGamutResultSpace = 0x000C,
    cmNamedIndexedSpace = 0x0010,
    cmMCFiveSpace = 0x0011,
    cmMCSixSpace = 0x0012,
    cmMCSevenSpace = 0x0013,
    cmMCEightSpace = 0x0014,
    cmAlphaPmulSpace = 0x0040,
    cmAlphaSpace = 0x0080,
    cmRGBASpace = cmRGBSpace + cmAlphaSpace,
    cmGrayASpace = cmGraySpace + cmAlphaSpace,
    cmRGBAPmulSpace = cmRGBASpace + cmAlphaPmulSpace,
    cmGrayAPmulSpace = cmGrayASpace + cmAlphaPmulSpace
};
```

Constants

`cmNoSpace`

The ColorSync Manager does not use this constant.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBSpace`

An RGB color space composed of red, green, and blue components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmCMYKSpace`

A CMYK color space composed of cyan, magenta, yellow, and black. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmHSVSpace`

An HSV color space composed of hue, saturation, and value components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmHLSSpace`
An HLS color space composed of hue, lightness, and saturation components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmYXYSpace`
A Yxy color space composed of Y, x, and y components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZSpace`
An XYZ color space composed of X, Y, and Z components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLUVSpace`
An L*u*v* color space composed of L*, u*, and v* components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLABSpace`
An L*a*b* color space composed of L*, a*, b* components. A bitmap never uses this constant alone. Instead, this color space is always combined with a packing format describing the amount of storage per component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmReservedSpace1`
This field is reserved for use by QuickDraw GX.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGraySpace`
A luminance color space with a single component, gray.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmReservedSpace2`
This field is reserved for use by QuickDraw GX.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGamutResultSpace`
A color space for the resulting bitmap pointed to by the `resultBitMap` field of the function `CWMatchColors` (page 828). A bitmap never uses this constant alone. Instead, it uses the constant `cmGamutResult1Space`, which combines `cmGamutResultSpace` and `cmOneBitDirectPacking` to define a bitmap that is 1 bit deep.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmNamedIndexedSpace`
A named indexed color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCFiveSpace`
A five-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSixSpace`
A six-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSevenSpace`
A seven-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCEightSpace`
An eight-channel multichannel (HiFi) data color space.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmAlphaPmulSpace`
A premultiplied alpha channel component is added to the color value.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmAlphaSpace`
An alpha channel component is added to the color value.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBASpace`
An RGB color space composed of red, green, and blue color value components and an alpha channel component. ColorSync does not currently support bitmaps that use this constant alone. Instead, this constant indicates the presence of an alpha channel in combination with `cmLong8ColorPacking` to indicate 8-bit packing format and `cmAlphaFirstPacking` to indicate the position of the alpha channel as the first component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGrayASpace`
A luminance color space with two components, a gray component followed by an alpha channel component. Each component value is 16 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBAPmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGrayAPmulSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Discussion
The data type `CMBitmap` (page 882) defines a bitmap for an image whose colors can be matched with the function `CWMatchColors` (page 828) or color-checked with the function `CWCheckColors` (page 821).

The `space` field of the `CMBitmap` type definition identifies the color space in which the colors of the bitmap image are specified. A color space is characterized by a number of components or dimensions, with each component carrying a numeric value. These values together make up the color value. A color space also specifies the format in which the color value is stored. For bitmaps in which color values are packed, the `space` field of the `CMBitmap` data type holds a constant that defines the color space and the packing format.

For the `CWMatchBitmap` function to perform color matching successfully, the color space specified in the `CMBitmap` data type's `space` field must correspond to the color space specified in the profile's `dataColorSpace` field. The source bitmap and source profile values must match and the destination bitmap and destination profile values must match. For the `CWCheckBitMap` function to perform color checking successfully, the source profile's `dataColorSpace` field value and the `space` field value of the source bitmap must specify the same color space. These functions will execute successfully as long as the color spaces are the same without regard for the packing format specified by the bitmap.

This enumeration defines constants for abstract color spaces which, when combined with a packing format constant as described in "Color Packing for Color Spaces" (page 957), can be used in the `space` field of the `CMBitmap` structure. The combined constants are shown in "Color Space Constants With Packing Formats" (page 962).

Version Notes
The constants `cmRGBASpace` and `cmGrayASpace` were moved to this enum from "Color Space Constants With Packing Formats" (page 962) in ColorSync version 2.5.

Calibrator Name Prefix

Specify an interface for new ColorSync monitor calibrators (ColorSync 2.6 and greater)

```
enum {
    kCalibratorNamePrefix = 'cali'
};
```

Constants
`kCalibratorNamePrefix`
Available in Mac OS X v10.0 and later.

Declared in `CMCalibrator.h`.

Channel Encoding Format

Specify an encoding format for sRGB64.

```
enum {
    cmSRGB16ChannelEncoding = 0x00010000
};
```

Constants

`cmSRGB16ChannelEncoding`
Used for sRGB64 encoding (±3.12 format)
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Chromatic Adaptation Values

Specify a transformation to use for chromaric adaptation.

```
typedef UInt32 CMChromaticAdaptation;
enum {
    cmUseDefaultChromaticAdaptation = 0,
    cmLinearChromaticAdaptation = 1,
    cmVonKriesChromaticAdaptation = 2,
    cmBradfordChromaticAdaptation = 3
};
```

Constants

`cmUseDefaultChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

`cmLinearChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

`cmVonKriesChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

`cmBradfordChromaticAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `CMTypes.h`.

CMM Function Selectors

Define selectors used for component-based CMM functions.

```
enum {
    kCMMOpen = -1,
    kCMMClose = -2,
    kCMMGetInfo = -4,
    kNCMMInit = 6,
    kCMMMatchColors = 1,
    kCMMCheckColors = 2,
    kCMMValidateProfile = 8,
    kCMMMatchBitmap = 9,
    kCMMCheckBitmap = 10,
    kCMMConcatenateProfiles = 5,
    kCMMConcatInit = 7,
    kCMMNewLinkProfile = 16,
    kNCMMConcatInit = 18,
    kNCMMNewLinkProfile = 19,
    kCMMGetPS2ColorSpace = 11,
    kCMMGetPS2ColorRenderingIntent = 12,
    kCMMGetPS2ColorRendering = 13,
    kCMMGetPS2ColorRenderingVMSize = 17,
    kCMMFlattenProfile = 14,
    kCMMUnflattenProfile = 15,
    kCMMInit = 0,
    kCMMGetNamedColorInfo = 70,
    kCMMGetNamedColorValue = 71,
    kCMMGetIndNamedColorValue = 72,
    kCMMGetNamedColorIndex = 73,
    kCMMGetNamedColorName = 74,
    kCMMMatchPixMap = 3,
    kCMMCheckPixMap = 4
};
```

Constants

`kCMMOpen`

Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMClose`

Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetInfo`

Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kNCMMInit`

In response to this request code, your CMM should initialize any private data it will need for the color session and for subsequent requests from the calling application or driver. Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMMatchColors`

In response to this request code, your CMM should match the colors in the `myColors` parameter to the color gamut of the destination profile and replace the color-list color values with the matched colors. Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMCheckColors`

In response to this request code, your CMM should test the given list of colors in the `myColors` parameter against the gamut specified by the destination profile and report if the colors fall within a destination device's color gamut. For more information, see the function `CWCheckColors` (page 821). Required.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMValidateProfile`

In response to this request code, your CMM should test the profile whose reference is passed in the `prof` parameter to determine if the profile contains the minimum set of elements required for a profile of its type. For more information, see the function `CMValidateProfile` (page 818).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMMatchBitmap`

In response to this request code, your CMM must match the colors of the source image bitmap pointed to by the `bitmap` parameter to the gamut of the destination device using the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWMatchBitmap` (page 826).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMCheckBitmap`

In response to this request code, your CMM must check the colors of the source image bitmap pointed to by the `bitmap` parameter against the gamut of the destination device using the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWCheckBitmap` (page 819).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMConcatenateProfiles`

This request code is for backward compatibility with ColorSync 1.0.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMConcatInit`

In response to this request code, your CMM should initialize any private data your CMM will need for a color session involving the set of profiles specified by the profile array pointed to by the `profileSet` parameter. Your function should also initialize any additional private data needed in handling subsequent calls pertaining to this component instance. For more information, see the function `CWConcatColorWorld` (page 823).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMNewLinkProfile`

In response to this request code, your CMM must create a single device link profile of type `DeviceLink` that includes the profiles passed to you in the array pointed to by the `profileSet` parameter. For more information, see the function `CWNewLinkProfile` (page 830).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kNCMMConcatInit`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kNCMMNewLinkProfile`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorSpace`

In response to this request code, your CMM must obtain or derive the color space element data from the source profile whose reference is passed to your function in the `srcProf` parameter and pass the data to a low-level data-transfer function supplied by the calling application or device driver. For more information, see the function `CMGetPS2ColorSpace` (page 776).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorRenderingIntent`

In response to this request code, your CMM must obtain the color-rendering intent from the header of the source profile whose reference is passed to your function in the `srcProf` parameter and then pass the data to a low-level data-transfer function supplied by the calling application or device driver. For more information, see the function `CMGetPS2ColorRenderingIntent` (page 774).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorRendering`

In response to this request code, your CMM must obtain the rendering intent from the source profile's header and generate the color rendering dictionary (CRD) data from the destination profile, and then pass the data to a low-level data-transfer function supplied by the calling application or device driver. For more information, see the function `CMGetPS2ColorRendering` (page 773).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetPS2ColorRenderingVMSize`

In response to this request code, your CMM must obtain or assess the maximum virtual memory (VM) size of the color rendering dictionary (CRD) specified by the destination profile. You must return the size of the CRD for the rendering intent specified by the source profile. For more information, see the function `CMGetPS2ColorRenderingVMSize` (page 775).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMFlattenProfile`

In response to this request code, your CMM must extract the profile data from the profile to flatten, identified by the `prof` parameter, and pass the profile data to the function specified in the `proc` parameter. For more information, see the function `CMFlattenProfile` (page 748).

Changed in ColorSync 2.5: Starting with ColorSync version 2.5, the ColorSync Manager calls the function provided by the calling program directly, without going through the preferred, or any, CMM. Your CMM only needs to handle this request code for versions of ColorSync prior to version 2.5.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMUnflattenProfile`

In response to this request code, your CMM must create a temporary file in which to store the profile data you receive from the low-level data-transfer function supplied by the calling application or driver. Your function must return the file specification.

Changed in ColorSync 2.5: Starting with ColorSync version 2.5, the ColorSync Manager calls the function provided by the calling program directly, without going through the preferred, or any, CMM. Your CMM only needs to handle this request code for versions of ColorSync prior to version 2.5.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMInit`

This request code is provided for backward compatibility with ColorSync 1.0. A CMM that supports ColorSync 1.0 profiles should respond to this request code by initializing any private data required for the color-matching or gamut-checking session to be held as indicated by subsequent request codes. If your CMM supports only ColorSync 1.0 profiles or both ColorSync 1.0 profiles and ColorSync Manager version 2.x profiles, you must support this request code. If you support only ColorSync Manager version 2.x profiles, you should return an unimplemented error in response to this request code.

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetNamedColorInfo`

In response to this request code, your CMM extracts named color data from the profile whose reference is passed in the `srcProf` parameter. For more information, see the function `CMGetNamedColorInfo` (page 763).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetNamedColorValue`

In response to this request code, your CMM extracts device and profile connection space (PCS) color values for a specific color name from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetNamedColorValue` (page 764).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetIndNamedColorValue`

In response to this request code, your CMM extracts device and PCS color values for a specific named color index from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetIndNamedColorValue` (page 759).

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

`kCMMGetNamedColorIndex`
: In response to this request code, your CMM extracts a named color index for a specific color name from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetNamedColorIndex` (page 762).

  Available in Mac OS X v10.0 through Mac OS X v10.3.

  Declared in `CMMComponent.h`.

`kCMMGetNamedColorName`
: In response to this request code, your CMM extracts a named color name for a specific named color index from the profile whose reference is passed in the `prof` parameter. For more information, see the function `CMGetNamedColorName` (page 763).

  Available in Mac OS X v10.0 through Mac OS X v10.3.

  Declared in `CMMComponent.h`.

`kCMMMatchPixMap`
: In response to this request code, your CMM must match the colors of the pixel map image pointed to by the `myPixMap` parameter to the gamut of the destination device, replacing the original pixel colors with their corresponding colors as specified in the data color space of the destination device's color gamut. To perform the matching, you use the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWMatchPixMap` (page 829).

  Available in Mac OS X v10.0 through Mac OS X v10.3.

  Declared in `CMMComponent.h`.

`kCMMCheckPixMap`
: In response to this request code, your CMM must check the colors of the pixel map image pointed to by the `myPixMap` parameter against the gamut of the destination device to determine if the pixel colors are within the gamut of the destination device and report the results. To perform the check, you use the profiles specified by a previous `kNCMMInit`, `kCMMInit`, or `kCMMConcatInit` request to your CMM. For more information, see the function `CWCheckPixMap` (page 822).

  Available in Mac OS X v10.0 through Mac OS X v10.3.

  Declared in `CMMComponent.h`.

Discussion
Your CMM must respond to the ColorSync Manager required request codes. When a CMM receives a required request code from the ColorSync Manager, the CMM must determine the nature of the request, perform the appropriate processing, set an error code if necessary, and return an appropriate function result to the Component Manager. The required request codes are:

- `kNCMMInit`
- `kCMMMatchColors`
- `kCMMCheckColors`
- `kCMMInit`

Your CMM should respond to the rest of the ColorSync Manager request codes defined by this enumeration, but it is not required to do so.

Color Management Module Component Interface

Specify a CMM interface version.

```
enum {
    CMMInterfaceVersion = 1
};
```

Constants

`CMMInterfaceVersion`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMMComponent.h`.

Discussion

If your CMM supports the ColorSync Manager version 2.x, it should return the constant defined by the following enumeration when the Component Manager calls your CMM with the `kComponentVersionSelect` request code.

In response to the `kComponentVersionSelect` request code, a CMM should set its entry point function's result to the CMM version number. The high-order 16 bits represent the major version and the low-order 16 bits represent the minor version. The `CMMInterfaceVersion` constant represents the major version number.

A CMM that only supports ColorSync 1.0 returns 0 for the major version in response to the version request.

The `kComponentVersionSelect` request code is one of four required Component Manager requests your CMM must handle.

Color Packing for Color Spaces

Specify how color values are stored.

```
enum {
    cmNoColorPacking = 0x0000,
    cmWord5ColorPacking = 0x0500,
    cmWord565ColorPacking = 0x0600,
    cmLong8ColorPacking = 0x0800,
    cmLong10ColorPacking = 0x0A00,
    cmAlphaFirstPacking = 0x1000,
    cmOneBitDirectPacking = 0x0B00,
    cmAlphaLastPacking = 0x0000,
    cm8_8ColorPacking = 0x2800,
    cm16_8ColorPacking = 0x2000,
    cm24_8ColorPacking = 0x2100,
    cm32_8ColorPacking = cmLong8ColorPacking,
    cm40_8ColorPacking = 0x2200,
    cm48_8ColorPacking = 0x2300,
    cm56_8ColorPacking = 0x2400,
    cm64_8ColorPacking = 0x2500,
    cm32_16ColorPacking = 0x2600,
    cm48_16ColorPacking = 0x2900,
    cm64_16ColorPacking = 0x2A00,
    cm32_32ColorPacking = 0x2700,
    cmLittleEndianPacking = 0x4000,
    cmReverseChannelPacking = 0x8000
};
```

Constants

`cmNoColorPacking`

This constant is not used for ColorSync bitmaps.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmWord5ColorPacking`

The color values for three 5-bit color channels are stored consecutively in 16-bits, with the highest order bit unused.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmWord565ColorPacking`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLong8ColorPacking`

The color values for three or four 8-bit color channels are stored consecutively in a 32-bit long. For three channels, this constant is combined with either `cmAlphaFirstPacking` or `cmAlphaLastPacking` to indicate whether the unused eight bits are located at the beginning or end.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLong10ColorPacking`

The color values for three 10-bit color channels are stored consecutively in a 32-bit long, with the two highest order bits unused.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmAlphaFirstPacking`
An alpha channel is added to the color value as its first component.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmOneBitDirectPacking`
One bit is used as the pixel format. This storage format is used by the resulting bitmap pointed to by the `resultBitMap` field of the function `CWMatchColors` (page 828); the bitmap must be only 1 bit deep.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmAlphaLastPacking`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm8_8ColorPacking`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm16_8ColorPacking`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm24_8ColorPacking`
The color values for three 8-bit color channels are stored in consecutive bytes, for a total of 24 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm32_8ColorPacking`
The color values for four 8-bit color channels are stored in consecutive bytes, for a total of 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm40_8ColorPacking`
The color values for five 8-bit color channels are stored in consecutive bytes, for a total of 40 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm48_8ColorPacking`
The color values for six 8-bit color channels are stored in consecutive bytes, for a total of 48 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm56_8ColorPacking`
The color values for seven 8-bit color channels are stored in consecutive bytes, for a total of 56 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm64_8ColorPacking`
The color values for eight 8-bit color channels are stored in consecutive bytes, for a total of 64 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm32_16ColorPacking`
The color values for two 16-bit color channels are stored in a 32-bit word.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm48_16ColorPacking`
The color values for three 16-bit color channels are stored in 48 consecutive bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm64_16ColorPacking`
The color values for four 16-bit color channels are stored in 64 consecutive bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cm32_32ColorPacking`
The color value for a 32-bit color channel is stored in a 32-bit word.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLittleEndianPacking`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmReverseChannelPacking`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Discussion
The ColorSync bitmap data type `CMBitmap` (page 882) includes a field that identifies the color space in which the color values of the bitmap image are expressed. This enumeration defines the types of packing for a color space's storage format. The enumeration also defines an alpha channel that can be added as a component of a color value to define the degree of opacity or transparency of a color. These constants are combined with the constants described in "Abstract Color Space Constants" (page 946) to create values that identify a bitmap's color space. Your application does not specify color packing constants directly, but rather uses the combined constants, which are described in "Color Space Constants With Packing Formats" (page 962).

Version Notes
The constants `cm48_16ColorPacking` and `cm64_16ColorPacking` were added in ColorSync version 2.5.

Color Responses

Specify responses for ColorSync 1.0 specifications.

```
enum {
    cmGrayResponse = 0,
    cmRedResponse = 1,
    cmGreenResponse = 2,
    cmBlueResponse = 3,
    cmCyanResponse = 4,
    cmMagentaResponse = 5,
    cmYellowResponse = 6,
    cmUcrResponse = 7,
    cmBgResponse = 8,
    cmOnePlusLastResponse = 9
};
```

Constants

`cmGrayResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmRedResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmGreenResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmBlueResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmCyanResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmMagentaResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmYellowResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmUcrResponse`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMICCProfile.h`.

`cmBgResponse`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmOnePlusLastResponse`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

Color Space Constants With Packing Formats

Specifies bitmap spaces with a wide range of data formats appropriate for multiple platforms.

```
enum {
    cmGray8Space = cmGraySpace + cm8_8ColorPacking,
    cmGray16Space = cmGraySpace,
    cmGray16LSpace = cmGraySpace + cmLittleEndianPacking,
    cmGrayA16Space = cmGrayASpace + cm16_8ColorPacking,
    cmGrayA32Space = cmGrayASpace,
    cmGrayA32LSpace = cmGrayASpace + cmLittleEndianPacking,
    cmGrayA16PmulSpace = cmGrayAPmulSpace + cm16_8ColorPacking,
    cmGrayA32PmulSpace = cmGrayAPmulSpace,
    cmGrayA32LPmulSpace = cmGrayAPmulSpace + cmLittleEndianPacking,
    cmRGB16Space = cmRGBSpace + cmWord5ColorPacking,
    cmRGB16LSpace = cmRGBSpace + cmWord5ColorPacking + cmLittleEndianPacking,
    cmRGB565Space = cmRGBSpace + cmWord565ColorPacking,
    cmRGB565LSpace = cmRGBSpace + cmWord565ColorPacking + cmLittleEndianPacking,
    cmRGB24Space = cmRGBSpace + cm24_8ColorPacking,
    cmRGB32Space = cmRGBSpace + cm32_8ColorPacking,
    cmRGB48Space = cmRGBSpace + cm48_16ColorPacking,
    cmRGB48LSpace = cmRGBSpace + cm48_16ColorPacking + cmLittleEndianPacking,
    cmARGB32Space = cmRGBASpace + cm32_8ColorPacking + cmAlphaFirstPacking,
    cmARGB64Space = cmRGBASpace + cm64_16ColorPacking + cmAlphaFirstPacking,
    cmARGB64LSpace = cmRGBASpace + cm64_16ColorPacking + cmAlphaFirstPacking
+ cmLittleEndianPacking,
    cmRGBA32Space = cmRGBASpace + cm32_8ColorPacking + cmAlphaLastPacking,
    cmRGBA64Space = cmRGBASpace + cm64_16ColorPacking + cmAlphaLastPacking,
    cmRGBA64LSpace = cmRGBASpace + cm64_16ColorPacking + cmAlphaLastPacking
+ cmLittleEndianPacking,
   cmARGB32PmulSpace = cmRGBAPmulSpace + cm32_8ColorPacking + cmAlphaFirstPacking,
   cmARGB64PmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaFirstPacking,
   cmARGB64LPmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaFirstPacking
+ cmLittleEndianPacking,
    cmRGBA32PmulSpace = cmRGBAPmulSpace + cm32_8ColorPacking + cmAlphaLastPacking,
   cmRGBA64PmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaLastPacking,
   cmRGBA64LPmulSpace = cmRGBAPmulSpace + cm64_16ColorPacking + cmAlphaLastPacking
+ cmLittleEndianPacking,
    cmCMYK32Space = cmCMYKSpace + cm32_8ColorPacking,
    cmCMYK64Space = cmCMYKSpace + cm64_16ColorPacking,
    cmCMYK64LSpace = cmCMYKSpace + cm64_16ColorPacking + cmLittleEndianPacking,
    cmHSV32Space = cmHSVSpace + cmLong10ColorPacking,
    cmHLS32Space = cmHLSSpace + cmLong10ColorPacking,
    cmYXY32Space = cmYXYSpace + cmLong10ColorPacking,
    cmXYZ24Space = cmXYZSpace + cm24_8ColorPacking,
    cmXYZ32Space = cmXYZSpace + cmLong10ColorPacking,
    cmXYZ48Space = cmXYZSpace + cm48_16ColorPacking,
    cmXYZ48LSpace = cmXYZSpace + cm48_16ColorPacking + cmLittleEndianPacking,
    cmLUV32Space = cmLUVSpace + cmLong10ColorPacking,
    cmLAB24Space = cmLABSpace + cm24_8ColorPacking,
    cmLAB32Space = cmLABSpace + cmLong10ColorPacking,
    cmLAB48Space = cmLABSpace + cm48_16ColorPacking,
    cmLAB48LSpace = cmLABSpace + cm48_16ColorPacking + cmLittleEndianPacking,
    cmGamutResult1Space = cmOneBitDirectPacking + cmGamutResultSpace,
    cmNamedIndexed32Space = cm32_32ColorPacking + cmNamedIndexedSpace,
    cmNamedIndexed32LSpace = cm32_32ColorPacking + cmNamedIndexedSpace
+ cmLittleEndianPacking,
    cmMCFive8Space = cm40_8ColorPacking + cmMCFiveSpace,
    cmMCSix8Space = cm48_8ColorPacking + cmMCSixSpace,
    cmMCSeven8Space = cm56_8ColorPacking + cmMCSevenSpace,
    cmMCEight8Space = cm64_8ColorPacking + cmMCEightSpace
};
```

```
typedef UInt32 CMBitmapColorSpace;
```

Constants

`cmGray8Space`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGray16Space`
A luminance color space with a single 16-bit component, gray.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGray16LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA16Space`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32Space`
A luminance color space with two components, a gray component followed by an alpha channel component. Each component value is 16 bits.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA16PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmGrayA32LPmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB16Space`
An RGB color space composed of red, green, and blue components whose values are packed with 5 bits of storage per component. The storage size for a color value expressed in this color space is 16 bits, with the high-order bit not used.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB16LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB565Space`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGB565LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB24Space`

An RGB color space composed of red, green, and blue components whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 24 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB32Space`

An RGB color space composed of red, green, and blue components whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with bits 24-31 not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB48Space`

An RGB color space composed of red, green, and blue components whose values are packed with 16 bits of storage per component. The storage size for a color value expressed in this color space is 48 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGB48LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB32Space`

An RGB color space composed of red, green, and blue color value components preceded by an alpha channel component whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB64Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmARGB64LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA32Space`

An RGB color space composed of red, green, and blue color value components, followed by an alpha channel component. Values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA64Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmRGBA64LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmARGB32PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmARGB64PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmARGB64LPmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGBA32PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGBA64PmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmRGBA64LPmulSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmCMYK32Space`
A CMYK color space composed of cyan, magenta, yellow, and black components whose values are packed with 8 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmCMYK64Space`
A CMYK color space composed of cyan, magenta, yellow, and black components whose values are packed with 16 bits of storage per component. The storage size for a color value expressed in this color space is 64 bits.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmCMYK64LSpace`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmHSV32Space`
An HSV color space composed of hue, saturation, and value components whose values are packed with 10 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmHLS32Space`

An HLS color space composed of hue, lightness, and saturation components whose values are packed with 10 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmYXY32Space`

A Yxy color space composed of Y, x, and y components whose values are packed with 10 bits of storage per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ24Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ32Space`

An XYZ color space composed of X, Y, and Z components whose values are packed with 10 bits per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ48Space`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmXYZ48LSpace`

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLUV32Space`

An L*u*v* color space composed of L*, u*, and v* components whose values are packed with 10 bits per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB24Space`

An L*a*b* color space composed of L*, a*, and b* components whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 24 bits. The 8-bit unsigned a* and b* channels are interpreted numerically as ranging from -128.0 to approximately 128.0.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB32Space`
An L*a*b* color space composed of L*, a*, and b* components whose values are packed with 10 bits per component. The storage size for a color value expressed in this color space is 32 bits, with the high-order 2 bits not used. The 10-bit unsigned a* and b* channels are interpreted numerically as ranging from -128.0 to approximately 128.0.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB48Space`
An L*a*b* color space composed of L*, a*, and b* components whose values are packed with 16 bits per component. The storage size for a color value expressed in this color space is 48 bits. The 16-bit unsigned a* and b* channels are interpreted numerically as ranging from -128.0 to approximately 128.0.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmLAB48LSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmGamutResult1Space`
A gamut result color space for the resulting bitmap pointed to by the `resultBitMap` field of the function `CWMatchColors` (page 828), with 1-bit direct packing. A pixel in the returned bitmap with value 1 (displayed as black) indicates an out-of-gamut color, while a pixel value of 0 (white) indicates a color that is in gamut.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmNamedIndexed32Space`
A color space where each color is stored as a single 32-bit value, specifying an index into a named color space. The storage size for a color value expressed in this color space is 32 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmNamedIndexed32LSpace`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCFive8Space`
A five-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 40 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSix8Space`
A six-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 48 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCSeven8Space`

A seven-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 56 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmMCEight8Space`

A eight-channel multichannel (HiFi) data color space, whose values are packed with 8 bits per component. The storage size for a color value expressed in this color space is 64 bits.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Discussion

This enumeration defines constants for color spaces which can specify color values for a bitmap image. As a rule, these constants include a packing format, defined in "Color Packing for Color Spaces" (page 957). You can use these constants to set the `space` field of the `CMBitmap` type definition identifies the color space in which the colors of the bitmap image are specified, as described in "Abstract Color Space Constants" (page 946).

Version Notes

The constants `cmRGBASpace` and `cmGrayASpace` were moved to "Abstract Color Space Constants" (page 946) in ColorSync version 2.5.

The constants `cmGray16Space`, `cmGrayA32Space`, `cmRGB48Space`, `cmCMYK64Space`, and `cmLAB48Space` were added in ColorSync version 2.5.

Color Space Signatures

Define four-character-sequences associated with color spaces.

```
enum {
    cmXYZData = 'XYZ ',
    cmLabData = 'Lab ',
    cmLuvData = 'Luv ',
    cmYCbCrData = 'YCbr',
    cmYxyData = 'Yxy ',
    cmRGBData = 'RGB ',
    cmSRGBData = 'sRGB',
    cmGrayData = 'GRAY',
    cmHSVData = 'HSV ',
    cmHLSData = 'HLS ',
    cmCMYKData = 'CMYK',
    cmCMYData = 'CMY ',
    cmMCH5Data = 'MCH5',
    cmMCH6Data = 'MCH6',
    cmMCH7Data = 'MCH7',
    cmMCH8Data = 'MCH8',
    cm3CLRData = '3CLR',
    cm4CLRData = '4CLR',
    cm5CLRData = '5CLR',
    cm6CLRData = '6CLR',
    cm7CLRData = '7CLR',
    cm8CLRData = '8CLR',
    cm9CLRData = '9CLR',
    cm10CLRData = 'ACLR',
    cm11CLRData = 'BCLR',
    cm12CLRData = 'CCLR',
    cm13CLRData = 'DCLR',
    cm14CLRData = 'ECLR',
    cm15CLRData = 'FCLR',
    cmNamedData = 'NAME'
};
```

Constants

`cmXYZData`
The XYZ data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmLabData`
The L*a*b* data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmLuvData`
The L*u*v* data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmYCbCrData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmYxyData`
The Yxy data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmRGBData`
The RGB data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSRGBData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGrayData`
The Gray data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmHSVData`
The HSV data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmHLSData`
The HLS data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCMYKData`
The CMYK data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCMYData`
The CMY data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH5Data`
The five-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH6Data`
The six-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH7Data`
The seven-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMCH8Data`
The eight-channel multichannel (HiFi) data color space.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm3CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm4CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm5CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm6CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm7CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm8CLRData`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cm9CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm10CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm11CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm12CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm13CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm14CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cm15CLRData`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmNamedData`
Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion
A ColorSync profile header contains a `dataColorSpace` field that carries the signature of the data color space in which the color values in an image using the profile are expressed. This enumeration defines the signatures for the color spaces supported by ColorSync for version 2.x profiles.

Color Space Masks

Specify masks used for color spaces.

```
enum {
    cmColorSpaceSpaceMask = 0x0000003F,
    cmColorSpacePremulAlphaMask = 0x00000040,
    cmColorSpaceAlphaMask = 0x00000080,
    cmColorSpaceSpaceAndAlphaMask = 0x000000FF,
    cmColorSpacePackingMask = 0x0000FF00,
    cmColorSpaceEncodingMask = 0x000F0000,
    cmColorSpaceReservedMask = 0xFFF00000
};
```

Constants
`cmColorSpaceSpaceMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmColorSpacePremulAlphaMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmColorSpaceAlphaMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmColorSpaceSpaceAndAlphaMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmColorSpacePackingMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmColorSpaceEncodingMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmColorSpaceReservedMask`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

ColorSync Scripting AppleEvent Errorsl

Define ColorSync AppleEvent scripting errors.

```
enum {
    cmspInvalidImageFile = -4220,
    cmspInvalidImageSpace = -4221,
    cmspInvalidProfileEmbed = -4222,
    cmspInvalidProfileSource = -4223,
    cmspInvalidProfileDest = -4224,
    cmspInvalidProfileProof = -4225,
    cmspInvalidProfileLink = -4226
};
```

Constants

`cmspInvalidImageFile`
Plugin cannot handle this image file type

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidImageSpace`
Plugin cannot create an image file of this colorspace

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidProfileEmbed`
Specific invalid profile errors

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidProfileSource`
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidProfileDest`
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidProfileProof`
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

`cmspInvalidProfileLink`
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

Current Device Versions

Specify the current versions of the data structure containing information on registered devices.

```
enum {
    cmDeviceInfoVersion1 = 0x00010000,
    cmDeviceProfileInfoVersion1 = 0x00010000,
    cmDeviceProfileInfoVersion2 = 0x00020000
};
```

Constants

`cmDeviceInfoVersion1`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceProfileInfoVersion1`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceProfileInfoVersion2`
Available in Mac OS X v10.1 and later.
Declared in `CMDeviceIntegration.h`.

Current Info Versions

Specify current device and profile versions.

```
enum {
    cmCurrentDeviceInfoVersion = cmDeviceInfoVersion1,
    cmCurrentProfileInfoVersion = cmDeviceProfileInfoVersion1
};
```

Constants

`cmCurrentDeviceInfoVersion`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmCurrentProfileInfoVersion`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

Current Major Version Mask

Specifies the current major version number.

```
enum {
    cmProfileMajorVersionMask = 0xFF000000,
    cmCurrentProfileMajorVersion = 0x02000000
};
```

Constants

`cmProfileMajorVersionMask`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCurrentProfileMajorVersion`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Data Transfer Commands

Specify commands for caller-supplied ColorSync data transfer functions.

```
enum {
    cmOpenReadSpool = 1,
    cmOpenWriteSpool = 2,
    cmReadSpool = 3,
    cmWriteSpool = 4,
    cmCloseSpool = 5
};
```

Constants

`cmOpenReadSpool`
Directs the function to begin the process of reading data.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmOpenWriteSpool`
Directs the function to begin the process of writing data.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmReadSpool`
Directs the function to read the number of bytes specified by the `CMFlattenProcPtr` function's `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmWriteSpool`
Directs the function to write the number of bytes specified by the `CMFlattenProcPtr` function's `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmCloseSpool`
Directs the function to complete the data transfer.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Discussion
When your application calls the function `CMFlattenProfile` (page 748), any of the functions in the group "Accessing Profile Elements" (page 719), or the PostScript-related functions of type "Working With PostScript" (page 722), the selected CMM—or, for the `CMUnflattenProfile` function, the ColorSync Manager—calls the flatten function you supply to transform profile data. The call passes one of the command constants defined by this enumeration.

Your application provides a pointer to your ColorSync data transfer function as a parameter to the functions. The ColorSync Manager or the CMM calls your data transfer function, passing the command in the `command` parameter. For more information on the flatten function, see `CMFlattenProfile` (page 748).

Data Type Element Values

Specify a data type.

```
enum {
    cmAsciiData = 0,
    cmBinaryData = 1
};
```

Constants
`cmAsciiData`
ASCII data.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBinaryData`
Binary data.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Default CMM Signature

Specifies a signature for the default color management module supplied by Color Sync.

```
enum {
    kDefaultCMMSignature = 'appl'
};
```

Constants
`kDefaultCMMSignature`
Signature for the default CMM supplied with the ColorSync Manager.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Discussion
A color management module (CMM) uses profiles to convert and match a color in a given color space on a given device to or from another color space or device.

To specify the default CMM, set the `CMMType` field of the profile header to the default signature defined by the following enumeration. You use a structure of type `CM2Header` (page 875) for a ColorSync 2.x profile and a structure of type `CMHeader` (page 898) for a 1.0 profile header.

Default IDs

Specify default values for device and profile IDs.

```
enum {
    cmDefaultDeviceID = 0,
    cmDefaultProfileID = 0
};
```

Constants

`cmDefaultDeviceID`

Available in Mac OS X v10.0 and later.

Declared in `CMDeviceIntegration.h`.

`cmDefaultProfileID`

Available in Mac OS X v10.0 and later.

Declared in `CMDeviceIntegration.h`.

Discussion

Certain routines require a device ID or profile ID. In some cases, a "default ID" can be used.

Device Attribute Values for Version 2.x Profiles

Define masks your application can use to set or test bits in the `deviceAttributes` field of the `CM2Header` structure.

```
enum {
    cmReflectiveTransparentMask = 0x00000001,
    cmGlossyMatteMask = 0x00000002
};
```

Constants

`cmReflectiveTransparentMask`

Bit 0 of `deviceAttributes[1]` specifies whether the media is transparent or reflective. If it has the value 0, the media is reflective; if it has the value 1, the media is transparent. Use the `cmReflectiveTransparentMask` mask to set the transparent/reflective bit in `deviceAttributes[1]` or to clear all bits except the transparent/reflective bit.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmGlossyMatteMask`

Bit 1of `deviceAttributes[1]` specifies whether the media is glossy or matte. If it has the value 0, the media is glossy; if it has the value 1, the media is matte. Use the `cmGlossyMatteMask` mask to set the glossy/matte bit in `deviceAttributes[1]` or to clear all bits except the glossy/matte bit.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion

The ColorSync Manager defines the structure `CM2Header` (page 875) to represent the profile header for the version 2.x profile format defined by the ICC.The `deviceAttributes` field of the `CM2Header` structure is an array of two unsigned long values whose bits specify information about a profile. The ICC reserves the use of `deviceAttributes[1]` and has assigned values to bits 0 and 1. All the bits of `deviceAttributes[0]` are reserved for use by color management system (CMS) vendors.

Device Classes

Define constants to represent a variey of input and output devices.

```
enum {
    cmScannerDeviceClass = 'scnr',
    cmCameraDeviceClass = 'cmra',
    cmDisplayDeviceClass = 'mntr',
    cmPrinterDeviceClass = 'prtr',
    cmProofDeviceClass = 'pruf'
};
typedef OSType CMDeviceClass;
```

Constants

`cmScannerDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmCameraDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDisplayDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmPrinterDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmProofDeviceClass`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

Device and Media Attributes

Used to set or obtaind device or media attributes.

```
enum {
    cmReflective = 0,
    cmGlossy = 1
};
```

Constants

`cmReflective`
If the bit 0 of the associated mask is `0` then reflective media; if `1` then transparency media.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGlossy`
If the bit 1 of the associated mask is is `0` then glossy; if `1` then matte.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Device States

Specify device states.

```
enum {
    cmDeviceStateDefault = 0x00000000,
    cmDeviceStateOffline = 0x00000001,
    cmDeviceStateBusy = 0x00000002,
    cmDeviceStateForceNotify = 0x80000000,
    cmDeviceStateDeviceRsvdBits = 0x00FF0000,
    cmDeviceStateAppleRsvdBits = 0xFF00FFFF
};
```

Constants

`cmDeviceStateDefault`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateOffline`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateBusy`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateForceNotify`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateDeviceRsvdBits`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

`cmDeviceStateAppleRsvdBits`
Available in Mac OS X v10.0 and later.
Declared in `CMDeviceIntegration.h`.

Discussion

Specify possible values for device states accessible by the functions `CMGetDeviceState` and `CMSetDeviceState`.

Device Types

Specify a device type.

```
enum {
    cmMonitorDevice = 'mntr',
    cmScannerDevice = 'scnr',
    cmPrinterDevice = 'prtr'
};
```

Constants

`cmMonitorDevice`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmScannerDevice`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmPrinterDevice`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

Element Tags and Signatures for Version 1.0 Profiles

Define tags and signatures used for version 1.0 profiles.

```
enum {
    cmCS1ChromTag = 'chrm',
    cmCS1TRCTag = 'trc ',
    cmCS1NameTag = 'name',
    cmCS1CustTag = 'cust'
};
```

Constants

`cmCS1ChromTag`

The tag signature for the profile chromaticities tag whose element data specifies the XYZ chromaticities for the six primary and secondary colors (red, green, blue, cyan, magenta, and yellow).

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1TRCTag`

The tag signature for profile tonal response curve data for the associated device.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1NameTag`

The tag signature for the profile name string. This is an international string consisting of a Macintosh script code followed by a 63-byte text string identifying the profile.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1CustTag`
Private data for a custom CMM.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion
The ICC version 2.x profile format differs from the version 1.0 profile format, and ColorSync Manager routines for updating a profile and searching for profiles do not work with version 1.0 profiles. However, your application can use version 1.0 profiles with all other ColorSync routines. For example, you can open a version 1.0 profile using the function `CMOpenProfile` (page 790), obtain the version 1.0 profile header using the function `CMGetProfileHeader` (page 769), and access version 1.0 profile elements using the function `CMGetProfileElement` (page 768).

To make this possible, the ColorSync Manager includes support for the version 1.0 profile header structure and synthesizes tags to allow you to access four 1.0 elements outside the version 1.0 profile header. This enumeration defines these tags.

Embedded Profile Flags

Specify copyright-protection flag options,

```
enum {
    cmEmbeddedProfile = 0,
    cmEmbeddedUse = 1
};
```

Constants
`cmEmbeddedProfile`
0 is not embedded profile, 1 is embedded profile

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmEmbeddedUse`
0 is to use anywhere, 1 is to use as embedded profile only

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Embedded Profile Identifiers

Specify constants used when embedding picture comments.

```
enum {
    cmEmbedWholeProfile = 0x00000000,
    cmEmbedProfileIdentifier = 0x00000001
};
```

Constants

`cmEmbedWholeProfile`

When the `flags` parameter has the value `cmEmbedWholeProfile`, the `NCMUseProfileComment` function embeds the entire specified profile.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEmbedProfileIdentifier`

When the `flags` parameter has the value `cmEmbedProfileIdentifier`, the `NCMUseProfileComment` function embeds a profile identifier for the specified profile.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Discussion

The ColorSync Manager provides these constant declarations to use with the function `NCMUseProfileComment` (page 843) for embedding picture comments. You use these constants to set the `flags` parameter to indicate whether to embed an entire profile or just a profile identifier.

Flag Mask Definitions for Version 2.x Profiles

Define masks your application can use to set or test various bits in the `flags` field of the `CM2Header` structure.

```
enum {
    cmICCReservedFlagsMask = 0x0000FFFF,
    cmEmbeddedMask = 0x00000001,
    cmEmbeddedUseMask = 0x00000002,
    cmCMSReservedFlagsMask = 0xFFFF0000,
    cmQualityMask = 0x00030000,
    cmInterpolationMask = 0x00040000,
    cmGamutCheckingMask = 0x00080000
};
```

Constants

`cmICCReservedFlagsMask`

This mask provides access to bits 0 through 15 of the `flags` field, which are defined and reserved by the ICC. For more information, see the International Color Consortium Profile Format Specification, and the next two mask definitions.

To obtain a copy of the ICC specification, or to get other information about the ICC, visit the ICC Web site at http://www.color.org/.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmEmbeddedMask`

This mask provides access to bit 0 of the `flags` field, which specifies whether the profile is embedded. It has the value 1 if the profile is embedded, 0 if it is not.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmEmbeddedUseMask`

This mask provides access to bit 1 of the `flags` field, which specifies whether the profile can be used independently or can only be used as an embedded profile. It has the value 0 if the profile can be used anywhere, 1 if it must be embedded.

You should interpret the setting of this bit as an indication of copyright protection. If the profile developer set this bit to 1, you should use this profile as an embedded profile only and not copy the profile for your own purposes. The profile developer also specifies explicit copyright intention using the `cmCopyrightTag` profile tag (defined in the `CMICCProfile.h` header file).

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCMSReservedFlagsMask`

This mask provides access to bits 16 through 31 of the `flags` field, which are available for a color management system (CMS) vendor, such as ColorSync. ColorSync's default CMM uses bits 16 through 19 to provide hints for color matching, as described in the following three mask definitions. Other CMM vendors should follow the same conventions.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmQualityMask`

This mask provides access to bits 16 and 17 of the `flags` field, which specify the preferred quality and speed preferences for color matching. In general, the higher the quality the slower the speed. For example, best quality is slowest, but produces the highest quality result.

Bits 16 and 17 have the value 0 for normal quality, 1 for draft quality, and 2 for best quality. "Quality Flag Values for Version 2.x Profiles" (page 1011) describes the constants ColorSync defines to test or set these bits.

This feature is provided by the ColorSync Manager; it is not defined by the ICC profile specification.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmInterpolationMask`

This mask provides access to bit 18 of the `flags` field, which specifies whether to use interpolation in color matching. The value 0 specifies interpolation. The value 1 specifies table lookup without interpolation. Specifying lookup only improves speed but can reduce accuracy. You might use lookup only for a monitor profile, for example, when high resolution is not crucial.

This feature is provided by the ColorSync Manager; it is not defined by the ICC profile specification.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmGamutCheckingMask`

This mask provides access to bit 19 of the `flags` field. When you use a profile to create a color world, bit 19 specifies whether the color world should include information for gamut checking. It has the value 0 if the color world should include a gamut-checking table, 1 if gamut-checking information is not required. ColorSync can create a color world without a gamut table more quickly and in less space.

Many applications do not perform gamut checking, so they should set this bit to 1. However, if you call a color checking function such as `CWCheckColors` (page 821), or `CWMatchColors` (page 828), after setting a profile's gamut-checking bit so that the color world does not contain gamut information, these routines return the `cmCantGamutCheckError` error.

This feature is provided by the ColorSync Manager; it is not defined by the ICC profile specification.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion

The `flags` field of the structure `CM2Header` (page 875) is an unsigned long value whose bits specify information about a profile. The ICC reserves the use of bits 0 to 15 and has assigned values to bits 0 and 1. Bits 16 to 31 are reserved for use by color management system (CMS) vendors. ColorSync has assigned values to bits 16 through 19.

ICC Profile Versions

Specify IDD profile version numbers.

```
enum {
    cmICCProfileVersion4 = 0x04000000,
    cmICCProfileVersion2 = 0x02000000,
    cmICCProfileVersion21 = 0x02100000,
    cmCS2ProfileVersion = cmICCProfileVersion2,
    cmCS1ProfileVersion = 0x00000100
};
```

Constants

`cmICCProfileVersion4`

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmICCProfileVersion2`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmICCProfileVersion21`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS2ProfileVersion`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmCS1ProfileVersion`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Illuminant Measurement Endocings

Specify standard illuminate measurement encodings.

```
enum {
    cmIlluminantUnknown = 0x00000000,
    cmIlluminantD50 = 0x00000001,
    cmIlluminantD65 = 0x00000002,
    cmIlluminantD93 = 0x00000003,
    cmIlluminantF2 = 0x00000004,
    cmIlluminantD55 = 0x00000005,
    cmIlluminantA = 0x00000006,
    cmIlluminantEquiPower = 0x00000007,
    cmIlluminantF8 = 0x00000008
};
```

Constants

`cmIlluminantUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD50`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD65`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD93`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantF2`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantD55`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantA`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantEquiPower`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmIlluminantF8`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Macintosh 68K Trap Word

Specifies a 68K trap word for the Macintosh.

```
enum {
    cmTrap = 0xABEE
};
```

Constants

`cmTrap`

Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `CMApplication.h`.

Magic Cookie Number

Specifies a magic cookie number for anonymous file ID.

```
enum {
    cmMagicNumber = 'acsp'
};
```

Constants

`cmMagicNumber`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Match Flags Field

Specifies a profile to match.

```
enum {
    cmspFavorEmbeddedMask = 0x00000001
};
```

Constants

`cmspFavorEmbeddedMask`

If bit 0 is `0` then use `srcProf` profile; if `1` then use profile embedded in image if present.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMScriptingPlugin.h`.

Match Profiles 2.0

Defines matching flags for version 2.0 of the `CMSearchRecord.searchMask`.

```
enum {
    cmMatchAnyProfile = 0x00000000,
    cmMatchProfileCMMType = 0x00000001,
    cmMatchProfileClass = 0x00000002,
    cmMatchDataColorSpace = 0x00000004,
    cmMatchProfileConnectionSpace = 0x00000008,
    cmMatchManufacturer = 0x00000010,
    cmMatchModel = 0x00000020,
    cmMatchAttributes = 0x00000040,
    cmMatchProfileFlags = 0x00000080
};
```

Constants

`cmMatchAnyProfile`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileCMMType`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileClass`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchDataColorSpace`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileConnectionSpace`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchManufacturer`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchModel`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchAttributes`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchProfileFlags`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Match Profiles 1.0

Defines matching flags for version 1.0 of the `CMSearchRecord.searchMask`.

```
enum {
    cmMatchCMMType = 0x00000001,
    cmMatchApplProfileVersion = 0x00000002,
    cmMatchDataType = 0x00000004,
    cmMatchDeviceType = 0x00000008,
    cmMatchDeviceManufacturer = 0x00000010,
    cmMatchDeviceModel = 0x00000020,
    cmMatchDeviceAttributes = 0x00000040,
    cmMatchFlags = 0x00000080,
    cmMatchOptions = 0x00000100,
    cmMatchWhite = 0x00000200,
    cmMatchBlack = 0x00000400
};
```

Constants

`cmMatchCMMType`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchApplProfileVersion`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDataType`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDeviceType`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDeviceManufacturer`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmMatchDeviceModel`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchDeviceAttributes`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchFlags`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchOptions`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchWhite`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

`cmMatchBlack`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CMApplication.h`.

Maximum Path Size

Specifies the maximum length for a path name.

```
enum {
    CS_MAX_PATH = 256
};
```

Constants

`CS_MAX_PATH`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Measurement Flares

Specify measurement flare encodings.

```
enum {
    cmFlare0 = 0x00000000,
    cmFlare100 = 0x00000001
};
```

Constants

`cmFlare0`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmFlare100`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Measurment Geometries

Specify measurement geometry encodings.

```
enum {
    cmGeometryUnknown = 0x00000000,
    cmGeometry045or450 = 0x00000001,
    cmGeometry0dord0 = 0x00000002
};
```

Constants

`cmGeometryUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGeometry045or450`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGeometry0dord0`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Obsolete Color Response Values

Redefines obsolete color response values.

```
enum {
    grayResponse = cmGrayResponse,
    redResponse = cmRedResponse,
    greenResponse = cmGreenResponse,
    blueResponse = cmBlueResponse,
    cyanResponse = cmCyanResponse,
    magentaResponse = cmMagentaResponse,
    yellowResponse = cmYellowResponse,
    ucrResponse = cmUcrResponse,
    bgResponse = cmBgResponse,
    onePlusLastResponse = cmOnePlusLastResponse
};
```

Obsolete Color Space Signatures

Redefines obsolete color space signatures.

```
enum {
    rgbData = cmRGBData,
    cmykData = cmCMYKData,
    grayData = cmGrayData,
    xyzData = cmXYZData
};
```

Obsolete Device Type Names

Redefines obsolete device type names.

```
enum {
    monitorDevice = cmMonitorDevice,
    scannerDevice = cmScannerDevice,
    printerDevice = cmPrinterDevice
};
```

Parametric Types

Specify a parametric curve type enumeration,

```
enum {
    cmParametricType0 = 0,
    cmParametricType1 = 1,
    cmParametricType2 = 2,
    cmParametricType3 = 3,
    cmParametricType4 = 4
};
```

Constants

`cmParametricType0`

Y = X^gamma

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType1`
Y = (aX+b)^gamma [X>=-b/a], Y = 0 [X<-b/a]

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType2`
Y = (aX+b)^gamma + c [X>=-b/a], Y = c [X<-b/a]

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType3`
Y = (aX+b)^gamma [X>=d], Y = cX [X<d]

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmParametricType4`
Y = (aX+b)^gamma + e [X>=d], Y = cX+f [X<d]

Available in Mac OS X v10.2 and later.

Declared in `CMICCProfile.h`.

Platform Enumeration Values

Specify computer platforms.

```
enum {
    cmMacintosh = 'APPL',
    cmMicrosoft = 'MSFT',
    cmSolaris = 'SUNW',
    cmSiliconGraphics = 'SGI ',
    cmTaligent = 'TGNT'
};
```

Profile Iteration Values

Specify profiles to iterate.

```
enum {
    cmIterateFactoryDeviceProfiles = 0x00000001,
    cmIterateCustomDeviceProfiles = 0x00000002,
    cmIterateCurrentDeviceProfiles = 0x00000003,
    cmIterateAllDeviceProfiles = 0x00000004,
    cmIterateDeviceProfilesMask = 0x0000000F
};
```

Constants

`cmIterateFactoryDeviceProfiles`
Iterate profiles registered through the routine `CMSetDeviceFactoryProfiles`. To retrieve all factory profiles for all devices, use `cmIterateFactoryDeviceProfiles` as the flags value when calling `CMIterateDeviceProfiles`. I

Available in Mac OS X v10.0 and later.

Declared in `CMDeviceIntegration.h`.

`cmIterateCustomDeviceProfiles`
: Iterate profiles that are meant to take the place of the factory profiles, as a result of customization or calibration. To retrieve only custom profiles for all devices, use the `cmIterateCustomDeviceProfiles`, as the flags value when calling `CMIterateDeviceProfiles`.

  Available in Mac OS X v10.0 and later.

  Declared in `CMDeviceIntegration.h`.

`cmIterateCurrentDeviceProfiles`
: Iterate profiles registered through the routing `CMSetDeviceProfiles`. To get the profiles in use for all devices, use `cmIterateCurrentDeviceProfiles` as the flags value. This will replace the factory profiles with any overrides, yielding the currently used set.l

  Available in Mac OS X v10.0 and later.

  Declared in `CMDeviceIntegration.h`.

`cmIterateAllDeviceProfiles`
: Iterate all profiles, without replacement.

  Available in Mac OS X v10.1 and later.

  Declared in `CMDeviceIntegration.h`.

`cmIterateDeviceProfilesMask`
: Available in Mac OS X v10.0 and later.

  Declared in `CMDeviceIntegration.h`.

Discussion
These are possible values for flags passed to the function `CMIterateDeviceProfiles`.

Profile Location Sizes

Specify a location size.

```
enum {
    cmOriginalProfileLocationSize = 72,
    cmCurrentProfileLocationSize = 2 + CS_MAX_PATH
};
```

Constants
`cmOriginalProfileLocationSize`
: Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmCurrentProfileLocationSize`
: Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

Profile Options

Specify a rendering intent.

```
enum {
    cmPerceptualMatch = 0x0000,
    cmColorimetricMatch = 0x0001,
    cmSaturationMatch = 0x0002
};
```

Constants

`cmPerceptualMatch`

Default. For photographic images

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmColorimetricMatch`

Exact matching when possible

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmSaturationMatch`

For solid colors

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

PostScript Data Formats

Specify constants that indicate the format of PostScript data.

```
enum {
    cmPS7bit = 1,
    cmPS8bit = 2
};
```

Constants

`cmPS7bit`

The data is 7-bit safe—therefore the data could be in 7-bit ASCII encoding or in ASCII base-85 encoding.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmPS8bit`

The data is 8-bit safe—therefore the data could be in 7-bit or 8-bit ASCII encoding.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Discussion

The ColorSync Manager provides these constant declarations to specify the format of PostScript data.

Picture Comment Kinds

Specify picture comment kinds for profiles and color matching.

```
enum {
    cmBeginProfile = 220,
    cmEndProfile = 221,
    cmEnableMatching = 222,
    cmDisableMatching = 223,
    cmComment = 224
};
```

Constants

`cmBeginProfile`

Indicates the beginning of a version 1.0 profile to embed. (To start a 2.x profile, you use `cmComment`.)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEndProfile`

Signals end of the use of an embedded version 2.x or 1.0 profile.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEnableMatching`

Turns on color matching for the ColorSync Manager. Do not nest `cmEnableMatching` and `cmDisableMatching` pairs.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmDisableMatching`

Turns off color matching for the ColorSync Manager. Do not nest `cmEnableMatching` and `cmDisableMatching` pairs. After the ColorSync Manager encounters this comment, it ignores all ColorSync-related picture comments until it encounters the next `cmEnableMatching` picture comment. At that point, the most recently used profile is reinstated.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmComment`

Provides information about a 2.x embedded profile or embedded profile identifier reference. This picture comment is followed by a 4-byte selector identifying what follows. "Picture Comment Selectors" (page 997) describes the possible selectors.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Discussion

The ColorSync Manager defines five picture comment kinds. You use these comments to embed a profile identifier, begin or end use of an embedded profile, and enable or disable color matching within drawing code sent to an output device. The `PicComment` function's `kind` parameter specifies the kind of picture comment.

Use a picture comment of kind `cmEndProfile` to explicitly terminate use of the currently effective embedded profile and begin use of the system profile. Otherwise, the currently effective profile remains in effect, leading to unexpected results if another picture follows that is meant to use the system profile and so is not preceded by a profile.

Picture Comment Selectors

Specify selectors to use in piture comments.

```
enum {
    cmBeginProfileSel = 0,
    cmContinueProfileSel = 1,
    cmEndProfileSel = 2,
    cmProfileIdentifierSel = 3
};
```

Constants

`cmBeginProfileSel`

Identifies the beginning of version 2.x profile data. The amount of profile data you can specify is limited to 32K minus 4 bytes for the selector.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmContinueProfileSel`

Identifies the continuation of version 2.x profile data. The amount of profile data you can specify is limited to 32K minus 4 bytes for the selector. You can use this selector repeatedly until all the profile data is embedded.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmEndProfileSel`

Signals the end of version 2.x profile data—no more data follows. Even if the amount of profile data embedded does not exceed 32K minus 4 bytes for the selector and your application did not use `cmContinueProfileSel`, you must terminate the process with `cmEndProfileSel`. Note that this selector has a behavior that is different from the `cmEndProfile` picture comment described in "Picture Comment Kinds" (page 995).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmProfileIdentifierSel`

Identifies the inclusion of profile identifier data. For information on embedding a profile identifier, see the function `NCMUseProfileComment` (page 843). For information on the format of profile identifier data, see `CMProfileIdentifier` (page 921).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

Discussion
To embed a version 2.x profile or profile identifier reference in a picture destined for display on another system or on a device such as a printer, your application uses the QuickDraw `PicComment` function. The ColorSync Manager provides the function `NCMUseProfileComment` (page 843) to embed picture comments. You specify a picture comment `kind` value of `cmComment` and a 4-byte selector describing the data in the picture comment.

Because a profile may exceed QuickDraw's 32 KB size limit for a picture comment, your application can use an ordered series of picture comments to embed a large profile.

You can also embed a profile identifier reference in a picture. The profile identifier may refer to a previously embedded profile, so that you do not have to embed the entire profile again, or it may refer to a profile stored on disk. When you embed a profile identifier, you can change certain values for the referred-to profile, including the quality flags and rendering intent. For more information on profile identifiers, see `CMProfileIdentifier` (page 921).

This enumeration defines the 4-byte selector values your application uses to identify the beginning and continuation of profile data and to signal the end of it.

Profile Access Procedures

Specify operations used to access profiles.

```
enum {
    cmOpenReadAccess = 1,
    cmOpenWriteAccess = 2,
    cmReadAccess = 3,
    cmWriteAccess = 4,
    cmCloseAccess = 5,
    cmCreateNewAccess = 6,
    cmAbortWriteAccess = 7,
    cmBeginAccess = 8,
    cmEndAccess = 9
};
```

Constants

`cmOpenReadAccess`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmOpenWriteAccess`
Open the profile for writing. The total size of the profile is specified in the `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmReadAccess`
Read the number of bytes specified by the `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmWriteAccess`
Write the number of bytes specified by the `size` parameter.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmCloseAccess`
: Close the profile for reading or writing.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmCreateNewAccess`
: Create a new data stream for the profile.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmAbortWriteAccess`
: Cancel the current write attempt.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmBeginAccess`
: Begin the process of procedural access. This is always the first operation constant passed to the access procedure. If the call is successful, the `cmEndAccess` operation is guaranteed to be the last call to the procedure.

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

`cmEndAccess`
: End the process of procedural access. This is always the last operation constant passed to the access procedure (unless the `cmBeginAccess` call failed).

  Available in Mac OS X v10.0 and later.

  Declared in `CMApplication.h`.

Discussion
When your application calls the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, or `CMNewLinkProfile` functions, it can supply the ColorSync Manager with a profile location structure of type `CMProcedureLocation` (page 919) to specify a procedure that provides access to a profile. The ColorSync Manager calls your procedure when the profile is created, initialized, opened, read, updated, or closed. The profile access procedure declaration is described in `CMProfileAccessProcPtr` (page 862).

When the ColorSync Manager calls your profile access procedure, it passes one of these constants in the `command` parameter to specify an operation. Your procedure must be able to respond to each of these constants.

Profile Classes

Specify profile class enumerations.

```
enum {
    cmInputClass = 'scnr',
    cmDisplayClass = 'mntr',
    cmOutputClass = 'prtr',
    cmLinkClass = 'link',
    cmAbstractClass = 'abst',
    cmColorSpaceClass = 'spac',
    cmNamedColorClass = 'nmcl'
};
```

Constants

`cmInputClass`
An input device profile defined for a scanner.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmDisplayClass`
A display device profile defined for a monitor.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmOutputClass`
An output device profile defined for a printer.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmLinkClass`
A device link profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmAbstractClass`
An abstract profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmColorSpaceClass`
A color space profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNamedColorClass`
A named color space profile.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Discussion

The ColorSync Manager supports seven classes, or types, of profiles.

A profile creator specifies the profile class in the profile header's `profileClass` field. For a description of the profile header, see `CM2Header` (page 875). This enumeration defines the profile class signatures.

Profile Concatenation Values

Specify values to use when concatnating profiles.

```
enum {
    kNoTransform = 0,
    kUseAtoB = 1,
    kUseBtoA = 2,
    kUseBtoB = 3,
    kDeviceToPCS = kUseAtoB,
    kPCSToDevice = kUseBtoA,
    kPCSToPCS = kUseBtoB,
    kUseProfileIntent = 0xFFFFFFFF
};
```

Constants

`kNoTransform`
Not used.
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseAtoB`
Use 'A2B*' tag from this profile or equivalent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseBtoA`
Use 'B2A*' tag from this profile or equivalent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseBtoB`
Use 'pre*' tag from this profile or equivalent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kDeviceToPCS`
Device Dependent to Device Independent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kPCSToDevice`
Device Independent to Device Dependent
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kPCSToPCS`
Independent, through device's gamut
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`kUseProfileIntent`
For renderingIntent in NCMConcatProfileSpec

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Profile Flags

Define flags that control native matchign and caching.

```
enum {
    cmNativeMatchingPreferred = 0x00000001,
    cmTurnOffCache = 0x00000002
};
```

Constants

`cmNativeMatchingPreferred`
Default to native not preferred

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

`cmTurnOffCache`
Default to turn on CMM cache

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMICCProfile.h`.

Profile Iteration Constants

Define an interation version.

```
enum {
    cmProfileIterateDataVersion1 = 0x00010000,
    cmProfileIterateDataVersion2 = 0x00020000,
    cmProfileIterateDataVersion3 = 0x00030000
};
```

Constants

`cmProfileIterateDataVersion1`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmProfileIterateDataVersion2`
Added makeAndModel

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmProfileIterateDataVersion3`
Added MD5 digest

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Profile Location Type

Defines profile location kinds.

```
enum {
    cmNoProfileBase = 0,
    cmFileBasedProfile = 1,
    cmHandleBasedProfile = 2,
    cmPtrBasedProfile = 3,
    cmProcedureBasedProfile = 4,
    cmPathBasedProfile = 5,
    cmBufferBasedProfile = 6
};
```

Constants

`cmNoProfileBase`
The profile is temporary. It will not persist in memory after its use for a color session. You can specify this type of profile location with the `CMNewProfile` and the `CMCopyProfile` functions.

Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmFileBasedProfile`
The profile is stored in a disk-file and the `CMProfLoc` union of type `CMProfLoc` (page 928) holds a structure of type `CMFileLocation` (page 896) identifying the profile file. You can specify this type of profile location with the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, and `CMNewLinkProfile` functions.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmHandleBasedProfile`
The profile is stored in relocatable memory and the `CMProfLoc` union of type `CMProfLoc` (page 928) holds a handle to the profile in a structure of type `CMHandleLocation` (page 898). You can specify this type of profile location with the `CMOpenProfile`, `CMNewProfile`, and `CMCopyProfile` functions.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmPtrBasedProfile`
The profile is stored in nonrelocatable memory and the `CMProfLoc` union of type `CMProfLoc` (page 928) holds a pointer to the profile in a structure of type `CMPtrLocation` (page 929). You can specify this type of profile location with the `CMOpenProfile` function only.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmProcedureBasedProfile`
The profile is in an arbitrary location, accessed through a procedure supplied by you. The `CMProfLoc` union of type `CMProfLoc` (page 928) holds a universal procedure pointer to your profile access procedure in a structure of type `CMProcedureLocation` (page 919). You can specify this type of profile location with the `CMOpenProfile`, `CMNewProfile`, `CMCopyProfile`, and `CMNewLinkProfile` functions. For a description of an application-supplied profile access procedure, see `CMProfileAccessProcPtr` (page 862).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CMApplication.h`.

`cmPathBasedProfile`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

`cmBufferBasedProfile`
Available in Mac OS X v10.0 and later.

Declared in `CMApplication.h`.

Discussion
Your application specifies the location for a profile using a profile location structure of type `CMProfileLocation` (page 924). A ColorSync profile that you open or create is typically stored in one of the following locations:

- In a disk file. The `u` field (a union) of the profile location data structure contains a file specification for a profile that is disk-file based. This is the most common way to store a ColorSync profile.
- In relocatable memory. The `u` field of the profile location data structure contains a handle specification for a profile that is stored in a handle.
- In nonrelocatable memory. The `u` field of the profile location data structure contains a pointer specification for a profile that is pointer based.
- In an arbitrary location, accessed by a procedure you provide. The `u` field of the profile location data structure contains a universal procedure pointer to your access procedure, as well as a pointer that may point to data associated with your procedure.

Additionally, your application can create a new or duplicate temporary profile. For example, you can use a temporary profile for a color-matching session and the profile is not saved after the session. For this case, the ColorSync Manager allows you to specify the profile location as having no specific location.

You use a pointer to a data structure of type `CMProfileLocation` to identify a profile's location when your application calls

- the `CMOpenProfile` function to obtain a reference to a profile
- the `CMNewProfile`, `CWNewLinkProfile`, or `CMCopyProfile` functions to create a new profile
- the `CMGetProfileLocation` function to get the location of an existing profile

Your application identifies the type of data the `CMProfileLocation u` field holds—a file specification, a handle, and so on—in the `CMProfileLocation` structure's `locType` field. You use the constants defined by this enumeration to identify the location type.

Public Tags

Specify tag values available for public use.

```
enum {
    cmAToB0Tag = 'A2B0',
    cmAToB1Tag = 'A2B1',
    cmAToB2Tag = 'A2B2',
    cmBlueColorantTag = 'bXYZ',
    cmBlueTRCTag = 'bTRC',
    cmBToA0Tag = 'B2A0',
    cmBToA1Tag = 'B2A1',
    cmBToA2Tag = 'B2A2',
    cmCalibrationDateTimeTag = 'calt',
    cmChromaticAdaptationTag = 'chad',
    cmCharTargetTag = 'targ',
    cmCopyrightTag = 'cprt',
    cmDeviceMfgDescTag = 'dmnd',
    cmDeviceModelDescTag = 'dmdd',
    cmGamutTag = 'gamt',
    cmGrayTRCTag = 'kTRC',
    cmGreenColorantTag = 'gXYZ',
    cmGreenTRCTag = 'gTRC',
    cmLuminanceTag = 'lumi',
    cmMeasurementTag = 'meas',
    cmMediaBlackPointTag = 'bkpt',
    cmMediaWhitePointTag = 'wtpt',
    cmNamedColorTag = 'ncol',
    cmNamedColor2Tag = 'ncl2',
    cmPreview0Tag = 'pre0',
    cmPreview1Tag = 'pre1',
    cmPreview2Tag = 'pre2',
    cmProfileDescriptionTag = 'desc',
    cmProfileSequenceDescTag = 'pseq',
    cmPS2CRD0Tag = 'psd0',
    cmPS2CRD1Tag = 'psd1',
    cmPS2CRD2Tag = 'psd2',
    cmPS2CRD3Tag = 'psd3',
    cmPS2CSATag = 'ps2s',
    cmPS2RenderingIntentTag = 'ps2i',
    cmRedColorantTag = 'rXYZ',
    cmRedTRCTag = 'rTRC',
    cmScreeningDescTag = 'scrd',
    cmScreeningTag = 'scrn',
    cmTechnologyTag = 'tech',
    cmUcrBgTag = 'bfd ',
    cmViewingConditionsDescTag = 'vued',
    cmViewingConditionsTag = 'view'
};
```

Constants

`cmAToB0Tag`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmAToB1Tag`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmAToB2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBlueColorantTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBlueTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBToA0Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBToA1Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmBToA2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCalibrationDateTimeTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmChromaticAdaptationTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCharTargetTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmCopyrightTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmDeviceMfgDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmDeviceModelDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGamutTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGrayTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGreenColorantTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmGreenTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmLuminanceTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMeasurementTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMediaBlackPointTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMediaWhitePointTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNamedColorTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNamedColor2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPreview0Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPreview1Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPreview2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmProfileDescriptionTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmProfileSequenceDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD0Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD1Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD2Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CRD3Tag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2CSATag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmPS2RenderingIntentTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmRedColorantTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmRedTRCTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmScreeningDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmScreeningTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmUcrBgTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmViewingConditionsDescTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmViewingConditionsTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Public Type Signatures

Specify signatures for public types.

```
enum {
    cmSigCrdInfoType = 'crdi',
    cmSigCurveType = 'curv',
    cmSigDataType = 'data',
    cmSigDateTimeType = 'dtim',
    cmSigLut16Type = 'mft2',
    cmSigLut8Type = 'mft1',
    cmSigMeasurementType = 'meas',
    cmSigMultiFunctA2BType = 'mAB ',
    cmSigMultiFunctB2AType = 'mBA ',
    cmSigNamedColorType = 'ncol',
    cmSigNamedColor2Type = 'ncl2',
    cmSigParametricCurveType = 'para',
    cmSigProfileDescriptionType = 'desc',
    cmSigProfileSequenceDescType = 'pseq',
    cmSigScreeningType = 'scrn',
    cmSigS15Fixed16Type = 'sf32',
    cmSigSignatureType = 'sig ',
    cmSigTextType = 'text',
    cmSigU16Fixed16Type = 'uf32',
    cmSigU1Fixed15Type = 'uf16',
    cmSigUInt8Type = 'ui08',
    cmSigUInt16Type = 'ui16',
    cmSigUInt32Type = 'ui32',
    cmSigUInt64Type = 'ui64',
    cmSigUcrBgType = 'bfd ',
    cmSigUnicodeTextType = 'utxt',
    cmSigViewingConditionsType = 'view',
    cmSigXYZType = 'XYZ '
};
```

Constants

`cmSigCrdInfoType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigCurveType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigDataType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigDateTimeType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigLut16Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigLut8Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigMeasurementType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigMultiFunctA2BType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigMultiFunctB2AType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigNamedColorType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigNamedColor2Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigParametricCurveType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigProfileDescriptionType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigProfileSequenceDescType`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmSigScreeningType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigS15Fixed16Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigSignatureType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigTextType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigU16Fixed16Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigU1Fixed15Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigUInt8Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigUInt16Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigUInt32Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigUInt64Type`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigUcrBgType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigUnicodeTextType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigViewingConditionsType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSigXYZType`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Quality Flag Values for Version 2.x Profiles

Define the possible values for the quality bits in the `flags` field of the `CM2Header` structure.

```
enum {
    cmNormalMode = 0,
    cmDraftMode = 1,
    cmBestMode = 2
};
```

Constants

`cmNormalMode`
This is the default setting. Normal mode indicates that the CMM should use its default method to compromise between performance and resource requirements.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmDraftMode`

Draft mode indicates that the CMM should sacrifice quality, if necessary, to minimize resource requirements. Note that the default CMM currently produces the same results for both normal and draft mode.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmBestMode`

Best mode indicates that the CMM should maximize resource usage to ensure the highest possible quality.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Discussion

To determine the value of the quality flag, you mask the `flags` field of the profile header with the `cmQualityMask` mask, right shift 16 bits, then compare the result to the enumerated constants shown here. For more information on the quality flag, see "Flag Mask Definitions for Version 2.x Profiles" (page 983).

When you start a color-matching session, ColorSync sends all involved profiles to the color management module (CMM). The CMM extracts the information it needs from the profiles and stores an internal representation in private memory. ColorSync's default CMM samples the input space and stores the results in a lookup table, a common technique that speeds up conversion for runtime applications. The size of the table is based on the quality flag setting in the source profile header. The setting of the quality flag can affect the memory requirements, accuracy, and speed of the color-matching session. In general, the higher the quality setting, the larger the lookup table, the more accurate the matching, and the slower the matching process. Note however, that the default CMM currently produces the same results for both normal and draft mode.

Rendering Intent Values for Version 2.x Profiles

Define the four possible values for the rendering intent bits of the `renderingIntent` field of the `CM2Header` structure.

```
enum {
    cmPerceptual = 0,
    cmRelativeColorimetric = 1,
    cmSaturation = 2,
    cmAbsoluteColorimetric = 3
};
```

Constants

`cmPerceptual`

All the colors of a given gamut can be scaled to fit within another gamut. This intent is best suited to realistic images, such as photographic images.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmRelativeColorimetric`

The colors that fall within the gamuts of both devices are left unchanged. This intent is best suited to logo images.

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSaturation`
: The relative saturation of colors is maintained from gamut to gamut. This intent is best suited to bar graphs and pie charts in which the actual color displayed is less important than its vividness.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

`cmAbsoluteColorimetric`
: This approach is based on a device-independent color space in which the result is an idealized print viewed on a ideal type of paper having a large dynamic range and color gamut.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

Discussion
The ColorSync Manager defines the structure `CM2Header` (page 875) to represent the profile header for the version 2.x profile format defined by the ICC. The `renderingIntent` field of the `CM2Header` structure is an unsigned long value whose bits specify information about a profile. The ICC reserves the use of bits 0 to 15 and has assigned values to bits 0 and 1. Bits 16 to 31 are reserved for use by color management system (CMS) vendors.

Rendering intent controls the approach a CMM uses to translate the colors of an image to the color gamut of a destination device. Your application can set a profile's rendering intent, for example, based on a user's choice of the preferred approach for rendering an image.

Because rendering intent is specified by the low two bits, and because no other bits are currently defined for this field, you can use the constants defined here to test or set the value of the entire field, without concern for possible information stored in other bits.

Screen Encoding Tags

Specify tags to use for screen encodings.

```
enum {
    cmPrtrDefaultScreens = 0,
    cmLinesPer = 1
};
```

Constants
`cmPrtrDefaultScreens`
: Use printer default screens; can have an associated value of `0` for `false` or `1` for `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

`cmLinesPer`
: Lines per unit; can have an associated value of `0` for lines per centimeter or `1` for lines per inch.

  Available in Mac OS X v10.0 and later.

  Declared in `CMICCProfile.h`.

Spot Function Values

Speicfy values for spot functions.

```
enum {
    cmSpotFunctionUnknown = 0,
    cmSpotFunctionDefault = 1,
    cmSpotFunctionRound = 2,
    cmSpotFunctionDiamond = 3,
    cmSpotFunctionEllipse = 4,
    cmSpotFunctionLine = 5,
    cmSpotFunctionSquare = 6,
    cmSpotFunctionCross = 7
};
```

Constants

`cmSpotFunctionUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionDefault`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionRound`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionDiamond`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionEllipse`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionLine`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionSquare`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmSpotFunctionCross`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Standard Oberver

Standard observer measurement type encodings.

```
enum {
    cmStdobsUnknown = 0x00000000,
    cmStdobs1931TwoDegrees = 0x00000001,
    cmStdobs1964TenDegrees = 0x00000002
};
```

Constants

`cmStdobsUnknown`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmStdobs1931TwoDegrees`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmStdobs1964TenDegrees`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Tag Type Information

Defines a constant for 2.0 tag type information.

```
enum {
    cmNumHeaderElements = 10
};
```

Constants

`cmNumHeaderElements`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Technology Tag Descriptions

Define descriptor tags for technologies.

```
enum {
    cmTechnologyDigitalCamera = 'dcam',
    cmTechnologyFilmScanner = 'fscn',
    cmTechnologyReflectiveScanner = 'rscn',
    cmTechnologyInkJetPrinter = 'ijet',
    cmTechnologyThermalWaxPrinter = 'twax',
    cmTechnologyElectrophotographicPrinter = 'epho',
    cmTechnologyElectrostaticPrinter = 'esta',
    cmTechnologyDyeSublimationPrinter = 'dsub',
    cmTechnologyPhotographicPaperPrinter = 'rpho',
    cmTechnologyFilmWriter = 'fprn',
    cmTechnologyVideoMonitor = 'vidm',
    cmTechnologyVideoCamera = 'vidc',
    cmTechnologyProjectionTelevision = 'pjtv',
    cmTechnologyCRTDisplay = 'CRT ',
    cmTechnologyPMDisplay = 'PMD ',
    cmTechnologyAMDisplay = 'AMD ',
    cmTechnologyPhotoCD = 'KPCD',
    cmTechnologyPhotoImageSetter = 'imgs',
    cmTechnologyGravure = 'grav',
    cmTechnologyOffsetLithography = 'offs',
    cmTechnologySilkscreen = 'silk',
    cmTechnologyFlexography = 'flex'
};
```

Constants

`cmTechnologyDigitalCamera`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyFilmScanner`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyReflectiveScanner`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyInkJetPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyThermalWaxPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyElectrophotographicPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyElectrostaticPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyDyeSublimationPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPhotographicPaperPrinter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyFilmWriter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyVideoMonitor`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyVideoCamera`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyProjectionTelevision`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyCRTDisplay`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPMDisplay`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyAMDisplay`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPhotoCD`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyPhotoImageSetter`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyGravure`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyOffsetLithography`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologySilkscreen`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmTechnologyFlexography`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Use Types

Specify use types.

```
enum {
    cmInputUse = 'inpt',
    cmOutputUse = 'outp',
    cmDisplayUse = 'dply',
    cmProofUse = 'pruf'
};
```

Constants

`cmInputUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmOutputUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmDisplayUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

`cmProofUse`
Available in Mac OS X v10.0 and later.
Declared in `CMApplication.h`.

Discussion
Used for the function `CMGetProfileByUse` and `SetDefaultProfileByUse`.

Video Card Gamma Storage Types

Specify data storage type constants.

```
enum {
    cmVideoCardGammaTableType = 0,
    cmVideoCardGammaFormulaType = 1
};
```

Constants

`cmVideoCardGammaTableType`
The video card gamma data is stored in a table format. See `CMVideoCardGammaTable` (page 941) for a description of the table format.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmVideoCardGammaFormulaType`
The video card gamma tag data is stored as a formula. See `CMVideoCardGammaFormula` (page 940) for a description of the formula format.
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

Discussion
A video card gamma profile tag can store gamma data either as a formula or as a table of values. You use a storage type constant to specify which data storage type the tag uses.

If the video card uses a different format than the format you specify (for example, the card uses data in table format and you supply data in formula format), ColorSync will adapt the data you supply to match the format the card expects.

Version Notes
Starting with version 2.5, ColorSync supports an optional profile tag for video card gamma. The tag specifies gamma information, stored either as a formula or in table format, to be loaded into the video card when the profile containing the tag is put into use. As of version 2.5, the only ColorSync function that attempts to take advantage of video card gamma data is `CMSetProfileByAVID` (page 808).

Video Card Gamma Tags

Specify video card gamma information.

```
enum {
    cmPS2CRDVMSizeTag = 'psvm',
    cmVideoCardGammaTag = 'vcgt',
    cmMakeAndModelTag = 'mmod',
    cmProfileDescriptionMLTag = 'dscm',
    cmNativeDisplayInfoTag = 'ndin'
};
```

Constants

`cmPS2CRDVMSizeTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmVideoCardGammaTag`
Constant for profile tag that specifies video card gamma information. When you create a tag to store video card gamma data in a profile, you use the `cmVideoCardGammaTag` constant to specify the tag.
Starting with version 2.5, ColorSync supports an optional profile tag for video card gamma. The tag specifies gamma information, stored either as a formula or in table format, to be loaded into the video card when the profile containing the tag is put into use. As of version 2.5, the only ColorSync function that attempts to take advantage of video card gamma data is `CMSetProfileByAVID` (page 808).
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmMakeAndModelTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmProfileDescriptionMLTag`
Available in Mac OS X v10.0 and later.
Declared in `CMICCProfile.h`.

`cmNativeDisplayInfoTag`
Available in Mac OS X v10.1 and later.
Declared in `CMICCProfile.h`.

Video Card Gamma Signatures

Specify signatures used for video card gamma information.

```
enum {
    cmSigPS2CRDVMSizeType = 'psvm',
    cmSigVideoCardGammaType = 'vcgt',
    cmSigMakeAndModelType = 'mmod',
    cmSigNativeDisplayInfoType = 'ndin',
    cmSigMultiLocalizedUniCodeType = 'mluc'
};
```

Constants

`cmSigPS2CRDVMSizeType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigVideoCardGammaType`

Constant that specifies video card gamma type signature in a video card gamma profile tag. That is, you use this constant to set the `typeDescriptor` field of the `CMVideoCardGammaType` (page 941) structure. There is currently only one type possible for a video card gamma tag.

Starting with version 2.5, ColorSync supports an optional profile tag for video card gamma. The tag specifies gamma information, stored either as a formula or in table format, to be loaded into the video card when the profile containing the tag is put into use. As of version 2.5, the only ColorSync function that attempts to take advantage of video card gamma data is `CMSetProfileByAVID` (page 808).

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigMakeAndModelType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

`cmSigNativeDisplayInfoType`

Available in Mac OS X v10.1 and later.

Declared in `CMICCProfile.h`.

`cmSigMultiLocalizedUniCodeType`

Available in Mac OS X v10.0 and later.

Declared in `CMICCProfile.h`.

Result Codes

The most common result codes returned by ColorSync Manager are listed below.

Result Code	Value	Description
`noErr`	0	No error Available in Mac OS X v10.0 and later.
`cmProfileError`	-170	There is something wrong with the content of the profile Available in Mac OS X v10.0 and later.

Result Code	Value	Description
cmMethodError	-171	An error occurred during the CMM arbitration process that determines the CMM to use Available in Mac OS X v10.0 and later.
cmMethodNotFound	-175	CMM not present Available in Mac OS X v10.0 and later.
cmProfileNotFound	-176	Responder error Available in Mac OS X v10.0 and later.
cmProfilesIdentical	-177	Profiles are the same Available in Mac OS X v10.0 and later.
cmCantConcatenateError	-178	Profiles cannot be concatenated Available in Mac OS X v10.0 and later.
cmCantXYZ	-179	CMM does not handle XYZ color space Available in Mac OS X v10.0 and later.
cmCantDeleteProfile	-180	Responder error Available in Mac OS X v10.0 and later.
cmUnsupportedDataType	-181	Responder error Available in Mac OS X v10.0 and later.
cmNoCurrentProfile	-182	Responder error Available in Mac OS X v10.0 and later.
cmElementTagNotFound	-4200	The tag you specified is not in the specified profile Available in Mac OS X v10.0 and later.
cmIndexRangeErr	-4201	Tag index out of range Available in Mac OS X v10.0 and later.
cmCantDeleteElement	-4202	Cannot delete the specified profile element Available in Mac OS X v10.0 and later.
cmFatalProfileErr	-4203	Returned from File Manager while updating a profile file in response to CMUpdateProfile; profile content may be corrupted Available in Mac OS X v10.0 and later.
cmInvalidProfile	-4204	Profile reference is invalid or refers to an inappropriate profile Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`cmInvalidProfileLocation`	-4205	Operation not supported for this profile location Available in Mac OS X v10.0 and later.
`cmInvalidSearch`	-4206	Bad search handle Available in Mac OS X v10.0 and later.
`cmSearchError`	-4207	Internal error occurred during profile search Available in Mac OS X v10.0 and later.
`cmErrIncompatibleProfile`	-4208	Unspecified profile error Available in Mac OS X v10.0 and later.
`cmInvalidColorSpace`	-4209	Profile color space does not match bitmap type Available in Mac OS X v10.0 and later.
`cmInvalidSrcMap`	-4210	Source pixel map or bitmap was invalid Available in Mac OS X v10.0 and later.
`cmInvalidDstMap`	-4211	Destination pix/bit map was invalid Available in Mac OS X v10.0 and later.
`cmNoGDevicesError`	-4212	Begin matching or end matching—no graphics devices available Available in Mac OS X v10.0 and later.
`cmInvalidProfileComment`	-4213	Bad profile comment during `drawpicture` Available in Mac OS X v10.0 and later.
`cmRangeOverFlow`	-4214	One or more output color value overflows in color conversion; all input color values will be converted and the overflow will be clipped Available in Mac OS X v10.0 and later.
`cmCantCopyModifiedV1Profile`	-4215	It is illegal to copy version 1.0 profiles that have been modified Available in Mac OS X v10.0 and later.
`cmNamedColorNotFound`	-4216	The specified named color was not found in the specified profile Available in Mac OS X v10.0 and later.
`cmCantGamutCheckError`	-4217	Gamut checking not supported by this color world—that is, the color world does not contain a gamut table because it was built with gamut checking turned off Available in Mac OS X v10.0 and later.

Result Code	Value	Description
cmDeviceDBNotFoundErr	-4227	Preferences not found or loaded; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
cmDeviceAlreadyRegistered	-4228	Device already registered; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
cmDeviceNotRegistered	-4229	Device not found; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
cmDeviceProfilesNotFound	-4230	Profiles not found; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.
cmInternalCFErr	-4231	CoreFoundation failure; returned by a CM device integration routine. Available in Mac OS X v10.0 and later.

Dictionary Manager Reference (Not Recommended)

Framework: ApplicationServices/ApplicationServices.h

Declared in Dictionary.h

Overview

Important: The Dictionary Manager is deprecated as of Mac OS X v10.5 and is not available to 64-bit applications. Instead, use the APIs presented by Dictionary Services, which are described in *Dictionary Services Programming Guide* and *Dictionary Services Reference*.

The Dictionary Manager facilitates the use of dictionary files by such programs as spelling checkers and input methods. The Dictionary Manager separates dictionary data from the code that accesses the data.

The Dictionary Manager uses access method plug-ins to mediate access to the dictionary's data. The use of access-method plug-ins allows the Dictionary Manager to support a variety of data and does not require the internal structure of a dictionary to conform to a fixed format.

Dictionary Manager functions with the prefix "DCM" are Carbon-compliant. However, these functions are available only on systems that have the Japanese language kit installed.

Functions by Task

Obtaining the Version Number

`DCMLibraryVersion` (page 1055) Deprecated in Mac OS X v10.5
Obtains the version number of the Dictionary Manager.

Working With a List of Dictionaries

You can use the functions in this section to obtain a list of the dictionaries available on the system. You can create a list of dictionaries, count the available dictionaries, and obtain the dictionary ID for each available dictionary.

`DCMCountObjectIterator` (page 1030) Deprecated in Mac OS X v10.5
Obtains the number of dictionaries in a list.

`DCMCreateDictionaryIterator` (page 1033) Deprecated in Mac OS X v10.5
Obtains a list of the dictionaries available on the system.

DCMDisposeObjectIterator (page 1037) Deprecated in Mac OS X v10.5
Disposes of a iterator.

DCMIterateObject (page 1055) Deprecated in Mac OS X v10.5
Obtains the object ID for a dictionary from a list of available dictionaries.

DCMResetObjectIterator (page 1060) Deprecated in Mac OS X v10.5
Resets an iterator to the start of the dictionary list.

Obtaining Access Method Information

DCMCreateAccessMethodIterator (page 1032) Deprecated in Mac OS X v10.5
Obtains a list of the available access methods.

DCMGetAccessMethodIDFromName (page 1040) Deprecated in Mac OS X v10.5
Obtains the ID for access method.

Working With a Dictionary File

DCMCloseDictionary (page 1029) Deprecated in Mac OS X v10.5
Closes a dictionary.

DCMCompactDictionary (page 1030) Deprecated in Mac OS X v10.5
Compacts a dictionary.

DCMCountRecord (page 1031) Deprecated in Mac OS X v10.5
Return number of records in a dictionary.

DCMDeleteDictionary (page 1035) Deprecated in Mac OS X v10.5
Deletes a dictionary.

DCMDeriveNewDictionary (page 1036) Deprecated in Mac OS X v10.5
Create a new dictionary based on an existing dictionary.

DCMGetDictionaryIDFromFile (page 1041) Deprecated in Mac OS X v10.5
Obtains the ID associated with a dictionary file.

DCMGetDictionaryIDFromRef (page 1042) Deprecated in Mac OS X v10.5
Obtains the dictionary ID associated with a dictionary reference.

DCMGetFileFromDictionaryID (page 1049) Deprecated in Mac OS X v10.5
Obtains the file specification associated with a dictionary ID.

DCMNewDictionary (page 1056) Deprecated in Mac OS X v10.5
Creates a new dictionary.

DCMOpenDictionary (page 1057) Deprecated in Mac OS X v10.5
Opens a dictionary.

DCMRegisterDictionaryFile (page 1058) Deprecated in Mac OS X v10.5
Registers a dictionary.

DCMReorganizeDictionary (page 1059) Deprecated in Mac OS X v10.5
Reorganizes a dictionary.

DCMUnregisterDictionary (page 1062) Deprecated in Mac OS X v10.5
Unregisters a dictionary.

Changing Access Privileges

`DCMGetDictionaryWriteAccess` (page 1044) Deprecated in Mac OS X v10.5
Obtains write access for an open dictionary.

`DCMReleaseDictionaryWriteAccess` (page 1059) Deprecated in Mac OS X v10.5
Releases write access to a dictionary.

Getting and Setting Dictionary Properties

`DCMGetDictionaryProperty` (page 1042) Deprecated in Mac OS X v10.5
Obtains the data associated with a property tag.

`DCMGetDictionaryPropertyList` (page 1043) Deprecated in Mac OS X v10.5
Obtains a list of property tags from a dictionary.

`DCMSetDictionaryProperty` (page 1061) Deprecated in Mac OS X v10.5
Sets a property for a dictionary. Set the properties to a dictionary.

Working With Dictionary Records

`DCMAddRecord` (page 1028) Deprecated in Mac OS X v10.5
Add a new record to a dictionary.

`DCMCountRecordIterator` (page 1032) Deprecated in Mac OS X v10.5
Returns the number of records contained in a list of search results.

`DCMDeleteRecord` (page 1035) Deprecated in Mac OS X v10.5
Delete specified record from the dictionary.

`DCMDisposeRecordIterator` (page 1037) Deprecated in Mac OS X v10.5
Disposes of a list of search results.

`DCMFindRecords` (page 1038) Deprecated in Mac OS X v10.5
Obtains a list of dictionary records that meet specified criteria.

`DCMGetNextRecord` (page 1050) Deprecated in Mac OS X v10.5
Obtains the next specified record.

`DCMGetNthRecord` (page 1051) Deprecated in Mac OS X v10.5
Return records in a specified order within the dictionary.

`DCMGetPrevRecord` (page 1052) Deprecated in Mac OS X v10.5
Obtains the previous record. Return the record previous to the specified record.

`DCMGetRecordSequenceNumber` (page 1053) Deprecated in Mac OS X v10.5
Obtains the sequence number for the specified record in a dictionary.

`DCMIterateFoundRecord` (page 1053) Deprecated in Mac OS X v10.5
Retrieves one record from a list of search results.

Working With Fields in a Single Record

`DCMCreateFieldInfoRecord` (page 1033) Deprecated in Mac OS X v10.5
Creates a field information record.

DCMGetDictionaryFieldInfo (page 1040) Deprecated in Mac OS X v10.5
Obtains field information for a specified field in a dictionary record.

DCMGetFieldAttributes (page 1045) Deprecated in Mac OS X v10.5
Obtains the field attributes for a field information record.

DCMGetFieldData (page 1046) Deprecated in Mac OS X v10.5
Obtains data from one or more fields in a specified record.

DCMGetFieldDefaultData (page 1047) Deprecated in Mac OS X v10.5
Obtains default data for a field information record.

DCMGetFieldFindMethods (page 1047) Deprecated in Mac OS X v10.5
Obtains the search methods for a field information record.

DCMGetFieldMaxRecordSize (page 1048) Deprecated in Mac OS X v10.5
Obtains the maximum data size for a field in a dictionary record.

DCMGetFieldTagAndType (page 1048) Deprecated in Mac OS X v10.5
Obtains the field tag and type associated with a field information record.

DCMSetFieldData (page 1061) Deprecated in Mac OS X v10.5
Set the data to a specific field of a specified record.

Functions

DCMAddRecord

Add a new record to a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMAddRecord (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   Boolean checkOnly,
   const AEDesc *dataList,
   DCMUniqueID *newUniqueID
);
```

Parameters

dictionaryRef
A reference to the dictionary to which you want to add a record. You obtain a dictionary reference when you call the function DCMOpenDictionary (page 1057).

keyFieldTag
The field tag that specifies the record you want to add.

keySize
The size of the keyData parameter.

keyData
The key data of the record you want to add. You are responsible for allocating this buffer.

checkOnly
If true is specified, only a duplication check of records is carried out, and actual addition of records is not carried out.

dataList

A pointer to the data contained in the record you want to add. The `AEDesc` data structure contains two fields: A four-character code that specifies the type of data in the structure and an opaque storage type that points to the storage for the descriptor data. Each of the data structures in the `dataList` array specifies a field name (the four-character code) in the dictionary record and the data associated with that field. You must first call the function `DCMCreateFieldInfoRecord` (page 1033) to create this array of data structures. When you create the data list, you must include all of the fields specified by the masks `kDCMIndexedFieldMask`, `kDCMRequiredFieldMask`, and `kDCMIdentifyFieldMask` as attributes of the field properties of the applicable dictionary.

newUniqueID

On output, the unique ID of the added record. If the added record is a duplicate record, the function returns the result `dcmDupRecordErr` and the unique ID of the existing record is returned in the `newUniqueID` parameter.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
When `checkOnly` is specified as `true`, if duplicate records do not exist, `noErr` is returned, and the `newUniqueID` value is undefined. When `checkOnly` is specified as `false`, if the same record exists, only the field data within that record is overwritten, and the unique ID of that record is returned as `newUniqueID`, and `dcmDupRecordErr` is returned as the return value. If the same record does not exist, the record is added, and the uniqueID of the added record is returned as `newUniqueID`, and `noErr` is returned as the return value.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMCloseDictionary

Closes a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMCloseDictionary (
   DCMDictionaryRef dictionaryRef
);
```

Parameters

dictionaryRef

A reference to the dictionary you want to close. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMCompactDictionary

Compacts a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMCompactDictionary (
   DCMDictionaryID dictionaryID,
   DCMProgressFilterUPP progressProc,
   UInt32 userData
);
```

Parameters

dictionaryID
The ID of the dictionary you want to compact. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

progressProc
A universal procedure pointer (UPP) to your progress callback function. This callback is not supported.

userData
Data needed by your progress callback function.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
The function `DCMCompactDictionary` organizes the contents of a dictionary to reduce the size of the dictionary as much as possible. You cannot add additional records to a dictionary after you have compacted it unless you call the function `DCMReorganizeDictionary` (page 1059) to expand the capacity of the dictionary.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMCountObjectIterator

Obtains the number of dictionaries in a list. (Deprecated in Mac OS X v10.5.)

```
ItemCount DCMCountObjectIterator (
   DCMObjectIterator iterator
);
```

Parameters

iterator

The list of available dictionaries. You obtain this list by calling the function `DCMCreateDictionaryIterator` (page 1033) or `DCMCreateAccessMethodIterator` (page 1032).

Return Value

The number of dictionaries contained in the list specified by the `iterator` parameter.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMCountRecord

Return number of records in a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMCountRecord (
   DCMDictionaryID dictionaryID,
   ItemCount *count
);
```

Parameters

dictionaryID

The ID of dictionary whose records you want to count. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

count

On output, the number of records in the dictionary.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMCountRecordIterator

Returns the number of records contained in a list of search results. (Deprecated in Mac OS X v10.5.)

```
ItemCount DCMCountRecordIterator (
   DCMFoundRecordIterator recordIterator
);
```

Parameters

recordIterator

A reference to list of search results. You obtain a list of search results by calling the function DCMFindRecords (page 1038).

Return Value

The number of records in the list specified by the recordIterator parameter.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

Dictionary.h

DCMCreateAccessMethodIterator

Obtains a list of the available access methods. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMCreateAccessMethodIterator (
   DCMAccessMethodIterator *accessMethodIterator
);
```

Parameters

accessMethodIterator

On output, a list of access methods.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion

You can operate on the list of access methods created by the function DCMCreateAccessMethodIterator by calling the functions DCMCountObjectIterator (page 1030), DCMIterateObject (page 1055), DCMResetObjectIterator (page 1060), and DCMDisposeObjectIterator (page 1037). You use this function along with the function DCMGetAccessMethodIDFromName (page 1040) to obtain the accessMethodID that you supply to the function DCMNewDictionary (page 1056) to create a new dictionary.

Normally you should not need to call this function because the Dictionary Manager handles the mapping of a dictionary to its access method.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMCreateDictionaryIterator

Obtains a list of the dictionaries available on the system. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMCreateDictionaryIterator (
   DCMDictionaryIterator *dictionaryIterator
);
```

Parameters

`dictionaryIterator`

On output, a reference to the list of available dictionaries. You are responsible for disposing of this list when you no longer need it.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
The function `DCMCreateDictionaryIterator` scans the dictionary directories, registers the dictionaries found, and returns a list of dictionaries. In Mac OS X, the Dictionary Manager searches `(User's home)/Library/Dictionaries/`, `/Library/Dictionaries/`, and their subdirectories. In Mac OS 9, it searches the Extensions folder, Preferences folder, and their subfolders.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMCreateFieldInfoRecord

Creates a field information record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMCreateFieldInfoRecord (
   DescType fieldTag,
   DescType fieldType,
   ByteCount maxRecordSize,
   DCMFieldAttributes fieldAttributes,
   AEDesc *fieldDefaultData,
   ItemCount numberOfFindMethods,
   DCMFindMethod findMethods[],
   AEDesc *fieldInfoRecord
);
```

Parameters

fieldTag

The tag for the field you want to create.

fieldType

The data type of the field.

maxRecordSize

The maximum size of the data in the field.

fieldAttributes

The attributes associated with the field. See "Field Attributes" (page 1073) for more information.

fieldDefaultData

On input, points to the default data for the field.

numberOfFindMethods

The number of search methods associated with the field.

findMethods

On input, an array of search methods associated with the field. See "Search Methods" (page 1078) for more information.

fieldInfoRecord

On output, points to the field information record for the newly-created field.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion

You can add multiple fields in the form of an `AEDescList` data structure by repeatedly calling the function `DCMCreateFieldInfoRecord`. You use the field information record (`fieldInfoRecord`) when you call the function `DCMNewDictionary`. However, when calling the function for the first time, you must set the `descriptorType` of the `fieldInfoRecord` to `typeNull`.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMDeleteDictionary

Deletes a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMDeleteDictionary (
   DCMDictionaryID dictionaryID
);
```

Parameters

dictionaryID

The ID of dictionary you want to delete. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080). Returns the result `dcmDictionaryBusyErr` if another application has this dictionary open.

Discussion

The function `DCMDeleteDictionary` deletes the dictionary specified by the `dictionaryID` parameter.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMDeleteRecord

Delete specified record from the dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMDeleteRecord (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   DCMUniqueID uniqueID
);
```

Parameters

dictionaryRef

A reference to the dictionary from which you want to delete a record. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag

The field tag that specifies the record you want to delete.

keySize

The size of the `keyData` parameter.

keyData

The key data of the record you want to delete. You are responsible for allocating this buffer.

uniqueID
The unique ID of the record you want to delete.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMDeriveNewDictionary

Create a new dictionary based on an existing dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMDeriveNewDictionary (
   DCMDictionaryID srcDictionary,
   const FSSpec *newDictionaryFile,
   ScriptCode scriptTag,
   Boolean invisible,
   ItemCount recordCapacity,
   DCMDictionaryID *newDictionary
);
```

Parameters

srcDictionary
The ID of dictionary from which you want to derive a dictionary. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

newDictionaryFile
A pointer to an `FSSpec` structure that specifies the file name and location for the newly-created dictionary. This is an input parameter.

scriptTag
The script code of the file specified by the `newDictionaryFile` parameter.

invisible
A Boolean value that specifies whether the dictionary is available through the Dictionary Manager. Pass `true` if you do not want the dictionary to be available, `false` otherwise. If you set invisible to `true`, that dictionary is no longer seen by such functions as `DCMCreateDictionaryIterator`, so it becomes a dictionary that cannot be accessed from any application other than the application that created the dictionary.

recordCapacity
The number of records that can be stored in the dictionary. You can supply an approximate value if you do not know the exact number.

newDictionary
On output, points to the ID of the newly-created dictionary.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
The function `DCMDeriveNewDictionary` creates an new dictionary file based on an existing dictionary specified in the `srcDictionary` parameter, with the same field configuration and properties, but does not contain the data from the source. The new dictionary is created with the name and at the location specified by the `newDictionaryFile` parameter. The newly-created dictionary is read-write enabled even if the source dictionary is read-only.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMDisposeObjectIterator

Disposes of a iterator. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMDisposeObjectIterator (
   DCMObjectIterator iterator
);
```

Parameters

iterator
The list of available dictionaries that you want to dispose of.You obtain this list by calling the function `DCMCreateDictionaryIterator` (page 1033) or `DCMCreateAccessMethodIterator` (page 1032).

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
You must dispose of a dictionary iterator when you no longer need it by calling the function `DCMDisposeObjectIterator` (page 1037).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMDisposeRecordIterator

Disposes of a list of search results. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMDisposeRecordIterator (
   DCMFoundRecordIterator recordIterator
);
```

Parameters

recordIterator

A reference to the list of search results you want to dispose of. You obtain a list of search results by calling the function `DCMFindRecords` (page 1038).

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMFindRecords

Obtains a list of dictionary records that meet specified criteria. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMFindRecords (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   DCMFindMethod findMethod,
   ItemCount preFetchedDataNum,
   DCMFieldTag preFetchedData[],
   ItemCount skipCount,
   ItemCount maxRecordCount,
   DCMFoundRecordIterator *recordIterator
);
```

Parameters

dictionaryRef

A reference to the dictionary you want to search. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag

A tag that specifies the field to search through. See "Field Data Tags" (page 1074) for a list of the fields you can specify for an Apple Japanese dictionary.

keySize

The length of the keyword specified by the `keyData` parameter.

keyData

The string for which you want to search.

findMethod

The search method to use. See "Search Methods" (page 1078) for a description of the methods you can supply.

preFetchedDataNum

The number of items in the `preFetchedData` array.

preFetchedData

An array of the tags obtained during the search. See "Field Data Tags" (page 1074) for a list of the field tags that can be obtained for an Apple Japanese dictionary.

skipCount

The number of records you want to skip during the search. You can use this value along with the `maxRecordCount` parameter to search through a dictionary in chunks. For example, if you want to obtain 10 matches at a time, the first time you search you should set the `skipCount` parameter to 0 and the `maxRecordCount` to 10. The second time you search you set `skipCount` to 10 and `maxRecordCount` to10. Each subsequent time you search, you increment `skipCount` by 10, keeping `maxRecordCount` set to10.

maxRecordCount

The maximum number of results to return. Pass 0 if you want all matching records returned. If the number of matching records is smaller than the maximum number of results to return, the search is terminated and the matching records returned in the `recordIterator` parameter. See the description of the `skipCount` parameter for information on how to use `maxRecordCount` to search through a dictionary in chunks.

recordIterator

On return, a reference to list of search results.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).If a match is not found, the function returns `dcmNoRecordErr`, and the `recordIterator` value is undefined.

Discussion

The function `DCMFindRecords` uses the specified key and search method to return a list of matching records. Search results are returned as a reference to a list of matching records. After you obtain this list, you can call the function `DCMGetFieldData` to retrieve each result in the list.

You can use the `preFetchedData` parameter to obtain a list of tags during the search. Data specified as pre-fetched are actually retrieved at the same time as the search key is retrieved, so you can specify data tags the need to be accessed fast and immediately after searching. In other words, you can avoid accessing and loading data that are not going to be used immediately by omitting those tags from the `preFetchedData` list. (You can retrieve those data later by calling the function `DCMGetFieldData` (page 1046).) For example: An application that searches pictures by date, displays the found titles in the list, and shows the picture only if the title is double-clicked, the key is "date", the pre-fetched data is "title", and the "picture" is retrieved later if needed by calling the function `DCMGetFieldData` (page 1046).

You pass the record iterator (`recordIterator`) as a parameter to the function DCM `DCMCountRecordIterator` (page 1032) to obtain the number of items in the list. You can obtain the individual items in the list by passing the record iterator to the function `DCMIterateFoundRecord` (page 1053). Your application is responsible for disposing of the record iterator when you no longer need it by calling the function `DCMDisposeRecordIterator` (page 1037).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetAccessMethodIDFromName

Obtains the ID for access method. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetAccessMethodIDFromName (
   ConstStr63Param accessMethodName,
   DCMAccessMethodID *accessMethodID
);
```

Parameters

accessMethodName
The name of access method whose ID you want to obtain.

accessMethodID
On output, the ID for the access method specified by the `accessMethodName` parameter.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
You can use this function to obtain the `accessMethodID` that you supply to the function `DCMNewDictionary` (page 1056) to create a new dictionary.

Normally you should not need to call this function because the Dictionary Manager handles the mapping of a dictionary to its access method.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetDictionaryFieldInfo

Obtains field information for a specified field in a dictionary record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetDictionaryFieldInfo (
   DCMDictionaryID dictionaryID,
   DCMFieldTag fieldTag,
   AEDesc *fieldInfoRecord
);
```

Parameters

dictionaryID

The ID of dictionary from which you want to obtain field information. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

fieldTag

The tag of the field whose field information you want to obtain. If you pass `'****'` (`typeWildCard`) as the field tag, all of the field information contained in the specified dictionary is returned in the `fieldInfoRecord` parameter.

fieldInfoRecord

On return, points to the field information for the specified field. You are responsible for disposing of this structure by calling the Apple Event Manager function `AEDisposeDesc`.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMGetDictionaryIDFromFile

Obtains the ID associated with a dictionary file. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetDictionaryIDFromFile (
   const FSSpec *fileRef,
   DCMDictionaryID *dictionaryID
);
```

Parameters

fileRef

The file specification for the dictionary whose ID you want to obtain.

dictionaryID

On output, points to the ID for the dictionary specified by the `fileRef` parameter.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080). If the file is not a dictionary, the result `dcmNotDictionaryErr` is returned. If the dictionary is not yet registered, the result `dcmBadDictionaryErr` is returned.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetDictionaryIDFromRef

Obtains the dictionary ID associated with a dictionary reference. (Deprecated in Mac OS X v10.5.)

```
DCMDictionaryID DCMGetDictionaryIDFromRef (
   DCMDictionaryRef dictionaryRef
);
```

Parameters

`dictionaryRef`

A reference to the dictionary whose ID you want to obtain. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

Return Value
On return, the ID of the dictionary specified by the `dictionaryRef` parameter. If `dictionaryRef` is invalid, the result `kDCMInvalidObjectID` is returned. See page for a description of the `DCMDictionaryID` data type.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetDictionaryProperty

Obtains the data associated with a property tag. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetDictionaryProperty (
   DCMDictionaryID dictionaryID,
   DCMFieldTag propertyTag,
   ByteCount maxPropertySize,
   ByteCount *actualSize,
   LogicalAddress propertyValue
);
```

Parameters

dictionaryID

The ID of dictionary whose property you want to obtain. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

propertyTag

The property tag whose data you want to obtain.

maxPropertySize

The size of the data specified by the `propertyValue` parameter.

actualSize

On output, the actual size of the data specified by the `propertyValue` parameter.

propertyValue

On output, points to the property data.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080). Returns the result `dcmBadPropertyErr` if the property tag does not exist.

Discussion

If you don't know the size of the property whose data you want to obtain, you need to call this function twice as follows:

- The first time you call the function `DCMGetProperty`, pass the dictionary ID, the property tag, `0` for the `maxPropertySize` parameter and `Null` for `propertyValue`. Then allocate a `propertyValue` buffer of the size returned by the `actualSize` parameter.
- The second time you call the function `DCMGetProperty`, pass the dictionary ID, the property tag, the correct size for the `maxPropertySize` parameter, and the `propertyValue` buffer of the appropriate size.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMGetDictionaryPropertyList

Obtains a list of property tags from a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetDictionaryPropertyList (
   DCMDictionaryID dictionaryID,
   ItemCount maxPropertyNum,
   ItemCount *numProperties,
   DCMFieldTag propertyTag[]
);
```

Parameters

dictionaryID

The ID of dictionary whose list of property tags you want to obtain. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

maxPropertyNum

The maximum number of property tags in the list.

numProperties

On output, the number of properties actually contained in the dictionary.

propertyTag

On output, an array of the property tags contained in the dictionary. You are responsible for allocating an array of the appropriate size.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion

The function `DCMGetDictionaryPropertyList` returns a list of property tags in the specified dictionary. You need to call this function twice. The first time you call the function to get the number of properties. Then you must allocate a `propertyTag` array of the appropriate size. You call the function a second time to get the actual list of tags.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMGetDictionaryWriteAccess

Obtains write access for an open dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetDictionaryWriteAccess (
   DCMDictionaryRef dictionaryRef,
   Duration timeOutDuration
);
```

Parameters

dictionaryRef

A reference to the dictionary for which you want to obtain write access. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

timeOutDuration

The maximum amount of time to wait for write access. This parameter is currently not used.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080). Returns the result `dcmPermissionErr` if the dictionary is read-only dictionary or if another application has write access.

Discussion
When you call the function `DCMOpenDictionary` (page 1057), the dictionary opens with read-only access. You must call the function `DCMGetDictionaryWriteAccess` to obtain write privileges. You can obtain write access only if a dictionary is already opened and no other application has write access to that dictionary. You should release write privileges as soon as you no longer need write access, by calling the function `DCMReleaseDictionaryWriteAccess` (page 1059).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetFieldAttributes

Obtains the field attributes for a field information record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetFieldAttributes (
   const AEDesc *fieldInfoRecord,
   DCMFieldAttributes *attributes
);
```

Parameters

fieldInfoRecord

On input, points to the field information record whose attributes you want to obtain.

attributes

On output, points to the attributes obtained from the field information record. See "Field Attributes" (page 1073)for more information.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetFieldData

Obtains data from one or more fields in a specified record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetFieldData (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   DCMUniqueID uniqueID,
   ItemCount numOfData,
   const DCMFieldTag dataTag[],
   AEDesc *dataList
);
```

Parameters

dictionaryRef

A reference to the dictionary that contains the field data you want to obtain. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag

A field tag that specifies the data you want to obtain.

keySize

The size of the `keyData` parameter.

keyData

The key data of the record you whose field you want to obtain. You are responsible for allocating this buffer.

uniqueID

The unique ID of the record whose field you want to obtain.

numOfData

The number of data fields tags in the `dataTag` array.

dataTag

A list of the data field to obtain.

dataList

On return, points to a list of obtained data. The data obtained is returned in t as an `AERecord` data structure. You can retrieve data from specific field using the Apple Event Manager function `AEGetKeyPtr`. You are responsible for disposing of the `dataList` array by calling the Apple Event Manager function `AEDisposeDesc`.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMGetFieldDefaultData

Obtains default data for a field information record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetFieldDefaultData (
   const AEDesc *fieldInfoRecord,
   DescType desiredType,
   AEDesc *fieldDefaultData
);
```

Parameters

fieldInfoRecord

On input, points to the field information record whose default data you want to obtain.

desiredType

The data type of the default data.

fieldDefaultData

On output, points to the default data obtained from the field information record.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMGetFieldFindMethods

Obtains the search methods for a field information record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetFieldFindMethods (
   const AEDesc *fieldInfoRecord,
   ItemCount findMethodsArrayMaxSize,
   DCMFindMethod findMethods[],
   ItemCount *actualNumberOfFindMethods
);
```

Parameters

fieldInfoRecord

On input, points to the field information record whose search methods you want to obtain.

findMethodsArrayMaxSize

The number of elements in the `findMethods` array.

findMethods

On output, an array of search methods obtained from the field information record. See "Search Methods" (page 1078) for more information.

actualNumberOfFindMethods

On output, the actual number of search methods obtained from the field information array.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetFieldMaxRecordSize

Obtains the maximum data size for a field in a dictionary record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetFieldMaxRecordSize (
   const AEDesc *fieldInfoRecord,
   ByteCount *maxRecordSize
);
```

Parameters
fieldInfoRecord
On input, points to the field information record whose maximum data size you want to obtain.
maxRecordSize
On output, points to the maximum data size obtained from the field information record.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetFieldTagAndType

Obtains the field tag and type associated with a field information record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetFieldTagAndType (
   const AEDesc *fieldInfoRecord,
   DCMFieldTag *fieldTag,
   DCMFieldType *fieldType
);
```

Parameters

fieldInfoRecord

On input, points to the field information record whose field tag and type you want to obtain.

fieldTag

On output, points to the field tag obtained from the field information record.

fieldType

On output, points to the field tag type obtained from the field information record.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMGetFileFromDictionaryID

Obtains the file specification associated with a dictionary ID. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetFileFromDictionaryID (
   DCMDictionaryID dictionaryID,
   FSSpec *fileRef
);
```

Parameters

dictionaryID

The ID of the dictionary whose file specification you want to obtain. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

fileRef

On output, points to the file specification for the dictionary specified by the `dictionaryID` parameter.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetNextRecord

Obtains the next specified record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetNextRecord (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   DCMUniqueID uniqueID,
   ByteCount maxKeySize,
   ByteCount *nextKeySize,
   LogicalAddress nextKeyData,
   DCMUniqueID *nextUniqueID
);
```

Parameters

dictionaryRef
A reference to the dictionary whose record you want to obtain. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag
The field tag that specifies the data you want to obtain.

keySize
The size of the `keyData` parameter. If you pass `0`, the first record in the dictionary is returned.

keyData
The key data of the reference record.

uniqueID
The unique ID of the reference record.

maxKeySize
The size of the buffer for the `nextKeyData` parameter.

nextKeySize
On output, the actual size of the buffer for the `nextKeyData` parameter.

nextKeyData
On output, points to the next key of the specified record. You must allocate this buffer.

nextUniqueID
On output, the unique ID of the found record.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetNthRecord

Return records in a specified order within the dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetNthRecord (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ItemCount serialNum,
   ByteCount maxKeySize,
   ByteCount *keySize,
   LogicalAddress keyData,
   DCMUniqueID *uniqueID
);
```

Parameters

dictionaryRef
A reference to the dictionary whose record you want to obtain. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag
The field tag that specifies the data you want to obtain.

serialNum
A value that specifies the location of the record within the dictionary. You can obtain this value by calling the function `DCMGetRecordSequenceNumber` (page 1053).

maxKeySize
The maximum size of the `keyData` parameter.

keySize
On output, the size of the `keyData` parameter.

keyData
The key data of the record you want to obtain. You are responsible for allocating this buffer.

uniqueID
On output, the unique ID of the found record.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMGetPrevRecord

Obtains the previous record. Return the record previous to the specified record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetPrevRecord (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   DCMUniqueID uniqueID,
   ByteCount maxKeySize,
   ByteCount *prevKeySize,
   LogicalAddress prevKeyData,
   DCMUniqueID *prevUniqueID
);
```

Parameters

dictionaryRef

A reference to the dictionary whose record you want to obtain. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag

The field tag that specifies the data you want to obtain.

keySize

The size of the `keyData` parameter. If you pass `0`, the last record in the dictionary is returned.

keyData

The key data of the reference record.

uniqueID

The unique ID of the record reference record.

maxKeySize

The size of the buffer for the `prevKeyData` parameter.

prevKeySize

On output, the actual size of the buffer for the `prevKeyData` parameter.

prevKeyData

On output, points to the previous key of the specified record. You must allocate this buffer.

prevUniqueID

On output, the unique ID of the found record.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMGetRecordSequenceNumber

Obtains the sequence number for the specified record in a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMGetRecordSequenceNumber (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   DCMUniqueID uniqueID,
   ItemCount *sequenceNum
);
```

Parameters

dictionaryRef

A reference to the dictionary whose record sequence number you want to obtain. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag

The field tag that specifies the data you want to obtain.

keySize

The size of the `keyData` parameter.

keyData

The key data of the record whose sequence number you want to obtain.

uniqueID

The unique ID of the record whose sequence number you want to obtain.

sequenceNum

On output, a value that specifies the order of the record in the dictionary. The first record in a dictionary has the value 1. Subsequent records are numbered sequentially.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMIterateFoundRecord

Retrieves one record from a list of search results. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMIterateFoundRecord (
   DCMFoundRecordIterator recordIterator,
   ByteCount maxKeySize,
   ByteCount *actualKeySize,
   LogicalAddress keyData,
   DCMUniqueID *uniqueID,
   AEDesc *dataList
);
```

Parameters

recordIterator

A reference to list of search results. You obtain a list of search results by calling the function DCMFindRecords (page 1038).

maxKeySize

The size of the keyData parameter.

actualKeySize

On output, the actual size of the buffer needed for the data specified by the keyData parameter.

keyData

On output, the key of the retrieved data.

uniqueID

On output, the unique ID of the retrieved record. This value is guaranteed to be unique among records with the same key data; but it is not unique among all records in the dictionary. You can use the unique ID in conjunction with the retrieved key data to specify individual records within the dictionary when you call the functions DCMGetNextRecord (page 1050), DCMGetPrevRecord (page 1052), DCMGetRecordSequenceNumber (page 1053), DCMDeleteRecord (page 1035), DCMGetFieldData (page 1046), and DCMSetFieldData (page 1061).

dataList

On output, the data associated with the key specified by the keyData parameter.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080). The function returns the result dcmIterationCompleteErr when the final record is retrieved.

Discussion

The function DCMIterateFoundRecord retrieves one record from a list of search results referenced by a record iterator. You can retrieve all of the records referenced by a record iterator by repeatedly calling the DCMIterateFoundRecord function.

The data associated with the fields specified in the preFetchedData parameter is returned to dataList in the form of an AERecord. (See DCMFindRecords (page 1038) for more information on pre-fetched data). It is possible to retrieve data by specifying the field tag and data type, and use the AEGetKeyPtr and so forth of the Apple Event Manager. You application is responsible for disposing of dataList by calling the Apple Event Manager function AEDisposeDesc.

Only pre-fetched data can be retrieved here since these data are already retrieved. Other data can be retrieved later by calling the function DCMGetFieldData (page 1046).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMIterateObject

Obtains the object ID for a dictionary from a list of available dictionaries. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMIterateObject (
   DCMObjectIterator iterator,
   DCMObjectID *objectID
);
```

Parameters

iterator

The list of available dictionaries. You obtain this list by calling the function `DCMCreateDictionaryIterator` (page 1033) or `DCMCreateAccessMethodIterator` (page 1032).

objectID

On output, the object ID of the dictionary.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
The first time you call the function `DCMIterateObject` it retrieves the object ID of the first dictionary in the list. The next time you call the function, it retrieves the object ID of the next dictionary in the list. You can obtain all object IDs by repeatedly calling this function. If you call the function after you obtain the object ID for the last dictionary, the function returns the result `dcmIterationCompleteErr`.

You can reset the iterator to the first dictionary in the list by calling the function `DCMResetObjectIterator` (page 1060). When you no longer need the iterator, you must dispose of it by calling the function `DCMDisposeObjectIterator` (page 1037).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMLibraryVersion

Obtains the version number of the Dictionary Manager. (Deprecated in Mac OS X v10.5.)

```
UInt32 DCMLibraryVersion (
   void
);
```

Parameters

Return Value
The library version number.

Discussion
The function `DCMLibraryVersion` returns the version of the installed Dictionary Manager in the same format as the `'vers'` resource. That is, the version number is returned in Binary-Coded Decimal (BCD) format in the high-order word, and the release stage information is returned in the low-order word. For example, the version 1.1.1 library returns 0x01118000.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMNewDictionary

Creates a new dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMNewDictionary (
   DCMAccessMethodID accessMethodID,
   const FSSpec *newDictionaryFile,
   ScriptCode scriptTag,
   const AEDesc *listOfFieldInfoRecords,
   Boolean invisible,
   ItemCount recordCapacity,
   DCMDictionaryID *newDictionary
);
```

Parameters

accessMethodID

The ID of access method to use for the dictionary. You can obtain an access method ID by calling the function `DCMGetAccessMethodIDFromName` (page 1040).

newDictionaryFile

On output, a pointer to an `FSSpec` structure that specifies the dictionary file to be created.

scriptTag

The code of the script system in which the filename of the dictionary file is to be displayed.

listOfFieldInfoRecords

A pointer to an array of `AEDesc` data structures. The `AEDesc` data structure contains two fields: A four-character code that specifies the type of data in the structure and an opaque storage type that points to the storage for the descriptor data. Each of the data structures in the `listOfFieldInfoRecords` array specifies a field name (the four-character code) in the dictionary record and the data associated with that field. You must first call the function `DCMCreateFieldInfoRecord` (page 1033) to create this array of data structures.

invisible

A Boolean value that specifies whether the dictionary is available through the Dictionary Manager. Pass `true` if you do not want the dictionary to be available, `false` otherwise. If you set invisible to `true`, that dictionary is no longer seen by such functions as `DCMCreateDictionaryIterator`, so it becomes a dictionary that cannot be accessed from any application other than the application that created the dictionary.

recordCapacity

The number of records that can be stored in the dictionary. You can supply an approximate value if you do not know the exact number.

newDictionary

On output, points to the ID of the newly-created dictionary.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion

The function `DCMNewDictionary` creates a new dictionary file and registers that dictionary with the Dictionary Manager. You need to specify the access method to use in creating the dictionary. The Dictionary Manager does not operate directly on a dictionary. Instead it accesses dictionaries using the access method specified for the dictionary. The access method mediates between the dictionary and the Dictionary Manager, so that the Dictionary Manager does not need to know anything about the physical format of the dictionary. As a result, the dictionary can have an free-form internal structure. You can also to use an existing dictionary as long as the dictionary has its own access method.

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMOpenDictionary

Opens a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMOpenDictionary (
   DCMDictionaryID dictionaryID,
   ByteCount protectKeySize,
   ConstLogicalAddress protectKey,
   DCMDictionaryRef *dictionaryRef
);
```

Parameters

dictionaryID

The ID of the dictionary you want to open. You obtain a dictionary ID when you register a dictionary by calling the function `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

protectKeySize

The size of the keyword specified by the `protectKey` parameter. This parameter is optional. Pass `0` if you do not plan to provide a password.

protectKey

The keyword to use when opening the dictionary. An access method can use the `protectKey` parameter as a password to restrict access to the dictionary. This parameter is optional. Pass `NULL` if you do not plan to provide a password.

dictionaryRef
On output, a reference to the opened dictionary.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
The function `DCMOpenDictionary` opens the dictionary specified by the dictionary ID and obtains a reference to the dictionary (`dictionaryRef`). You can pass this reference as a parameter to other Dictionary Manager functions to access the records in the dictionary.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMRegisterDictionaryFile

Registers a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMRegisterDictionaryFile (
   const FSSpec *dictionaryFile,
   DCMDictionaryID *dictionaryID
);
```

Parameters
dictionaryFile
The file specification for the dictionary you want to register.
dictionaryID
On output, points to the ID of the registered dictionary. A dictionary ID is not persistent across system restarts.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
You should use this function only when the target dictionary is not in default location (see`DCMCreateDictionaryIterator` (page 1033)). Otherwise, dictionaries should be already registered and you can obtain the `dictionaryID` by calling the function `DCMGetDictionaryIDFromFile` (page 1041) or `DCMCreateDictionaryIterator` (page 1033).

You can only use registered dictionaries. You pass the dictionary ID obtained from the function `DCMRegisterDictionaryFile` when you call other Dictionary Manager functions.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMReleaseDictionaryWriteAccess

Releases write access to a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMReleaseDictionaryWriteAccess (
   DCMDictionaryRef dictionaryRef,
   Boolean commitTransaction
);
```

Parameters

dictionaryRef

A reference to the dictionary for which you want to release write access. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

commitTransaction

A Boolean value that specifies whether or not to write changed to the dictionary. Pass `true` to write changes to the dictionary. Pass `false` to cancel changes. The dictionary must support transaction processing for this parameter to have an effect. This parameter is currently not used.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMReorganizeDictionary

Reorganizes a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMReorganizeDictionary (
   DCMDictionaryID dictionaryID,
   ItemCount extraCapacity,
   DCMProgressFilterUPP progressProc,
   UInt32 userData
);
```

Parameters

dictionaryID

The ID of dictionary you want to reorganize. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

extraCapacity
The number of additional records you want to add. This number can be approximate.

progressProc
A universal procedure pointer (UPP) to a progress callback function. This callback is not supported.

userData
Data needed by your progress callback function.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
The function `DCMReorganizeDictionary` reorganizes the contents of the dictionary specified by the `dictionaryID` parameter, expanding the dictionary to allow for the additional number of records specified by the `extraCapacity` parameter.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMResetObjectIterator

Resets an iterator to the start of the dictionary list. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMResetObjectIterator (
   DCMObjectIterator iterator
);
```

Parameters

iterator
The list of available dictionaries. You obtain this list by calling the function `DCMCreateDictionaryIterator` (page 1033) or `DCMCreateAccessMethodIterator` (page 1032).

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080).

Discussion
If you want to retrieve the object ID for a dictionary in the list, call the function `DCMIterateObject` (page 1055). When you no longer need the iterator, you must dispose of it by calling the function `DCMDisposeObjectIterator` (page 1037).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMSetDictionaryProperty

Sets a property for a dictionary. Set the properties to a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMSetDictionaryProperty (
   DCMDictionaryID dictionaryID,
   DCMFieldTag propertyTag,
   ByteCount propertySize,
   ConstLogicalAddress propertyValue
);
```

Parameters

dictionaryID
The ID of dictionary whose property you want to set. You obtain a dictionary ID when you call the functions `DCMRegisterDictionaryFile` (page 1058) or `DCMGetDictionaryIDFromFile` (page 1041).

propertyTag
The property tag whose data you want to set.

propertySize
The size of data pointed to by the `propertyValue` parameter.

propertyValue
A pointer to the property data to be set.

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080). Returns the result `dcmPermissionErr` if the property already exists and it is a read-only property.

Discussion
If the specified properties already exists and it is a writable property, the property data is replaced. If the property does not exist, it is created.

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMSetFieldData

Set the data to a specific field of a specified record. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMSetFieldData (
   DCMDictionaryRef dictionaryRef,
   DCMFieldTag keyFieldTag,
   ByteCount keySize,
   ConstLogicalAddress keyData,
   DCMUniqueID uniqueID,
   const AEDesc *dataList
);
```

Parameters

dictionaryRef

A reference to the dictionary that contains the field data you want to set. You obtain a dictionary reference when you call the function `DCMOpenDictionary` (page 1057).

keyFieldTag

Tag of applicable key field.

keySize

The size of the `keyData` parameter.

keyData

The key data of the record you whose field you want to set.

uniqueID

The unique ID of the record whose field you want to obtain.

dataList

A pointer to the list of data you want to set. The `AEDesc` data structure contains two fields: A four-character code that specifies the type of data in the structure and an opaque storage type that points to the storage for the descriptor data. Each of the data structures in the `dataList` array specifies a field name (the four-character code) in the dictionary record and the data associated with that field. You must first call the function `DCMCreateFieldInfoRecord` (page 1033) to create this array of data structures.

Return Value

A result code. See "Dictionary Manager Result Codes" (page 1080).

Availability

Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMUnregisterDictionary

Unregisters a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus DCMUnregisterDictionary (
   DCMDictionaryID dictionaryID
);
```

Parameters

dictionaryID

Return Value
A result code. See "Dictionary Manager Result Codes" (page 1080). Returns `dcmDictionaryBusyErr` if the dictionary is in use by another client.

Discussion
You should use this function only for dictionaries that you registered by calling the function `DCMRegisterDictionaryFile` (page 1058).

Availability
Available in CarbonLib 1.0 and later when running Japanese Mac OS 8.5 or later, or other Mac OS 8.5 or later with the Japanese Language Kit.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

Callbacks

DCMProgressFilterProcPtr

Displays the progress state of a reorganization or compaction operation.

```
typedef Boolean (*DCMProgressFilterProcPtr)
(
    Boolean determinateProcess,
    UInt16 percentageComplete,
    UInt32 callbackUD
);
```

If you name your function `MyDCMProgressFilterProc`, you would declare it like this:

```
Boolean DCMProgressFilterProcPtr (
    Boolean determinateProcess,
    UInt16 percentageComplete,
    UInt32 callbackUD
);
```

Discussion
You supply this callback as a parameter to the `DCMReorganizeDictionary` (page 1059) and `DCMCompactDictionary` (page 1030) functions.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Dictionary.h`

Data Types

DCMAccessMethodID

Represents an access method ID.

```
typedef DCMObjectID DCMAccessMethodID;
```

Discussion
See `DCMGetAccessMethodIDFromName` (page 1040).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMAccessMethodIterator

Represents a list of access methods.

```
typedef DCMObjectIterator DCMAccessMethodIterator;
```

Discussion
See `DCMCreateAccessMethodIterator` (page 1032).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMDictionaryHeader

Contains header information for a dictionary.

```
struct DCMDictionaryHeader {
    FourCharCode headerSignature;
    UInt32 headerVersion;
    ByteCount headerSize;
    Str63 accessMethod;
};
typedef struct DCMDictionaryHeader DCMDictionaryHeader;
```

Fields

`headerSignature`
: The header signature must be `'dict'`.

`headerVersion`
: The version of header information. The current version is specified by the constant `kDCMDictionaryHeaderVersion`.

`headerSize`
: The size of the header information. The current size is 76 bytes.

`accessMethod`
: The library name of the access method.

Discussion

The internal structure of dictionaries used by the Dictionary Manager can be broadly divided into three structures: dictionary header section, dictionary property section, and data record section. However, internal structures apart from the dictionary header section rely on the access method and can be freely defined, so you must be aware that this is strictly a structural model for dictionaries viewed from outside.

Dictionary files that are managed by the Dictionary Manager have a dictionary header at the start of its data fork in a defined format, and this header must contain a signature which indicates that it is a dictionary supported by the Dictionary Manager. The file must also contain basic information such as the name of the access method which can manage this dictionary. The dictionary header is the only information in the dictionary that can be accessed without the use of an access method.

The dictionary property section contains information about the dictionary as a whole, such as access restrictions (whether it is write enabled, whether its content can be downloaded). Properties are managed by tags, and it is possible to define and save different types of information.

The data record section contains registered data records. Each data record is further divided into fields," and these are managed using tags which represent the meaning of the data—for example, 'yomi' (read) and 'hins' (part of speech). At least one field in a record must be a key field. The key field is used as an index, and tag and key data for this field are used to find records. When searching through records, you can specify other field tags in addition to the key field, thereby enabling a variety of data to be obtained at once.

Each field in a record has field attributes that specify the field's data type, maximum data length, and default data The key data field also has an attribute that specifies the search methods it supports.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dictionary.h`

DCMDictionaryID

Represents the ID associated with a specific dictionary.

```
typedef DCMObjectID DCMDictionaryID;
```

Discussion
When you call the function `DCMRegisterDictionaryFile` (page 1058) to register a dictionary, the Dictionary Manager assigned a unique ID that you need to use in subsequent calls to the Dictionary Manager. The `DCMDictionaryID` value is not persistent across system restarts, so you must not save this value for future use. Each time your application launches or the system starts up you need to obtain the newly-assigned `DCMDictionaryID` value.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMDictionaryIterator

Represents a list of dictionaries.

```
typedef DCMObjectIterator DCMDictionaryIterator;
```

Discussion
See `DCMCreateDictionaryIterator` (page 1033).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMDictionaryRef

Represents a reference to a dictionary.

```
typedef DCMObjectRef DCMDictionaryRef;
```

Discussion
You can obtain a dictionary reference by calling the function `DCMOpenDictionary` (page 1057). You need this reference to operate on the dictionary.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMFieldTag

Represents a field inside a dictionary.

```
typedef DescType DCMFieldTag;
```

Discussion
A field tag is a 4-byte value used to identify a fields. A field must be unique within a dictionary. See "Field Data Tags" (page 1074) for more information.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMFieldType

Represents the data type of the data stored in a field.

```
typedef DescType DCMFieldType;
```

Discussion
A field tag is a 4-byte value used to specify the data type contained in a field. The basic definition is the data type defined by Apple Event Manager. "Field Data Types" (page 1074).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMFoundRecordIterator

Represents a reference to an opaque list of search results.

```
typedef struct OpaqueDCMFoundRecordIterator * DCMFoundRecordIterator;
```

Discussion
See `DCMFindRecords` (page 1038) for more information.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMObjectID

Represents a reference to an opaque dictionary ID.

```
typedef struct OpaqueDCMObjectID * DCMObjectID;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMObjectIterator

Defines a reference to an opaque dictionary object iterator.

```
typedef struct OpaqueDCMObjectIterator * DCMObjectIterator;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMObjectRef

Defines a reference to an opaque dictionary reference.

```
typedef struct OpaqueDCMObjectRef * DCMObjectRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMProgressFilterUPP

Defines a universal procedure pointer (UPP) to a progress filter callback.

```
typedef DCMProgressFilterProcPtr DCMProgressFilterUPP;
```

Discussion
For more information see `DCMProgressFilterProcPtr` (page 1063). You pass a progress filter callback UPP to the functions `DCMReorganizeDictionary` (page 1059) and `DCMCompactDictionary` (page 1030).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

DCMUniqueID

Represents the unique ID of a record in a dictionary.

```
typedef UInt32 DCMUniqueID;
```

Discussion
This ID is used in many functions. For example, see `DCMGetNextRecord` (page 1050),`DCMGetPrevRecord` (page 1052), and `DCMGetNthRecord` (page 1051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dictionary.h`

Constants

Access Method Features

Specify features associated with an access method.

```
enum {
    kDCMCanUseFileDictionaryMask = 0x00000001,
    kDCMCanUseMemoryDictionaryMask = 0x00000002,
    kDCMCanStreamDictionaryMask = 0x00000004,
    kDCMCanHaveMultipleIndexMask = 0x00000008,
    kDCMCanModifyDictionaryMask = 0x00000010,
    kDCMCanCreateDictionaryMask = 0x00000020,
    kDCMCanAddDictionaryFieldMask = 0x00000040,
    kDCMCanUseTransactionMask = 0x00000080
};
typedef OptionBits DCMAccessMethodFeature;
```

Constants

`kDCMCanUseFileDictionaryMask`
Specifies the file dictionary mask.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMCanUseMemoryDictionaryMask`
Specifies the memory dictionary mask
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMCanStreamDictionaryMask`
Specifies t the stream dictionary mask

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMCanHaveMultipleIndexMask`
Specifies the multiple index mask

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMCanModifyDictionaryMask`
Specifies the modify dictionary mask

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMCanCreateDictionaryMask`
Specifies the create dictionary mask

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMCanAddDictionaryFieldMask`
Specifies to use the add dictionary field mask

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMCanUseTransactionMask`
Specifies to use the use transaction dictionary mask

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Discussion
The Dictionary Manager does not operate directly on a dictionary. Instead it accesses dictionaries using the access method specified for the dictionary. The access method mediates between the dictionary and the Dictionary Manager, so that the Dictionary Manager does not need to know anything about the physical format of the dictionary. As a result, the dictionary can have an free-form internal structure. You can also to use an existing dictionary as long as the dictionary has its own access method.

Normally (except when you create a dictionary) you do not need to recognize the existence of the access method.

Dictionary Classes

Specify dictionary classes associated with the property `pDCMClass`.

```
enum {
    kDCMUserDictionaryClass = 0,
    kDCMSpecificDictionaryClass = 1,
    kDCMBasicDictionaryClass = 2
};
```

Constants

`kDCMUserDictionaryClass`
Indicates a user dictionary.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMSpecificDictionaryClass`
Indicates a specific dictionary.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMBasicDictionaryClass`
Indicates a basic dictionary.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

Dictionary Information Constants

Specify various data in a dictionary.

```
enum {
    kDictionaryFileType = 'dict',
    kDCMDictionaryHeaderSignature = 'dict',
    kDCMDictionaryHeaderVersion = 2
};
```

Constants

`kDictionaryFileType`
Specify the file type.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMDictionaryHeaderSignature`
Specify the header in a dictionary.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMDictionaryHeaderVersion`
Specify the header version.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Dictionary Properties

Specify standard dictionary properties.

```
enum {
    pDCMAccessMethod = 'amtd',
    pDCMPermission = 'perm',
    pDCMListing = 'list',
    pDCMMaintenance = 'mtnc',
    pDCMLocale = 'locl',
    pDCMClass = pClass,
    pDCMCopyright = 'info'
};
```

Constants

`pDCMAccessMethod`
Specifies an access method; the associated data type is `typeChar` and is read-only. This property is required.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`pDCMPermission`
Specifies a permission level; the associated data type is `typeUInt16`. This property is required.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`pDCMListing`
Specifies whether or not to allow data to be downloaded from the dictionary; the associated data type is `typeUInt16`. This property is optional.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`pDCMMaintenance`
This property is obsolete.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`pDCMLocale`
: Specifies a script code for the dictionary contents; the associated data type is `typeUInt32`. The default is `kLocaleIdentifierWildCard`. This property is optional.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`pDCMClass`
: Specifies a dictionary class; the associated data type is `typeUInt16`. The possible values are `kDCMUserDictionaryClass`, `kDCMSpecificDictionaryClass`), and `kDCMBasicDictionaryClass`. This property is optional.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`pDCMCopyright`
: Specifies copyright information; the associated data type is `typeChar` and is read-only.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Discussion
Of the properties listed here, the only property that must always be supported is the `pDCMPermission` property. If a existing property is set by calling the function `DCMSetDictionaryProperty` (page 1061) the property data is overwritten, except if the property is defined as read-only property. However, some read-write properties can be restricted so they can be modified only by an access method.

Functions that obtain data from a dictionary cannot be used on any dictionary whose `pDCMListing` property is set to `kDCMProhibitListing`. In this case, the function trying to obtain the data returns the result `dcmPermissionErr`. This means the dictionary data is protected because it is impossible to output the contents of the dictionary as text.

Field Attributes

Specify attributes for fields in a dictionary.

```
enum {
    kDCMIndexedFieldMask = 0x00000001,
    kDCMRequiredFieldMask = 0x00000002,
    kDCMIdentifyFieldMask = 0x00000004,
    kDCMFixedSizeFieldMask = 0x00000008,
    kDCMHiddenFieldMask = 0x80000000
};
typedef OptionBits DCMFieldAttributes;
```

Constants
`kDCMIndexedFieldMask`
: Specifies a key field that can be used in search.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMRequiredFieldMask`
Specifies an essential field in which data must always be provided when adding a record.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMIdentifyFieldMask`
Specifies a field that can be used to identify the same record.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMFixedSizeFieldMask`
Specifies a fixed-size field.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMHiddenFieldMask`
Specfies a hidden field that is not returned by the function that obtains the field list.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Discussion
The records contained in a dictionary have a variety of fields. For example, a record in a dictionary for kana-kanji conversion, typically has a "read" key field which stores hiragana character strings that are found, as well as a "part of speech" and a "notation" fields that refers to the search results. These respective fields also require attributes such as data type, maximum length of stored data, and fixed length/variable length.

When you create a new dictionary using the Dictionary Manager, you can specify the fields to include in the dictionary and the attributes associated with each of those field.

Field Data Tags

Specify field tags in an Apple Japanese dictionary.

```
enum {
    kDCMJapaneseYomiTag = 'yomi',
    kDCMJapaneseHyokiTag = 'hyok',
    kDCMJapaneseHinshiTag = 'hins',
    kDCMJapaneseWeightTag = 'hind',
    kDCMJapanesePhoneticTag = 'hton',
    kDCMJapaneseAccentTag = 'acnt',
    kDCMJapaneseOnKunReadingTag = 'OnKn',
    kDCMJapaneseFukugouInfoTag = 'fuku'
};
```

Field Data Types

Specify the data types for the values associated with field tags.

```
enum {
    kDCMJapaneseYomiType = typeUnicodeText,
    kDCMJapaneseHyokiType = typeUnicodeText,
    kDCMJapaneseHinshiType = 'hins',
    kDCMJapaneseWeightType = typeShortInteger,
    kDCMJapanesePhoneticType = typeUnicodeText,
    kDCMJapaneseAccentType = 'byte',
    kDCMJapaneseOnKunReadingType = typeUnicodeText,
    kDCMJapaneseFukugouInfoType = 'fuku'
};
```

Constants

`kDCMJapaneseYomiType`
Specifies the data type is Unicode text.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMJapaneseHyokiType`
Specifies the data type is Unicode text.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMJapaneseHinshiType`
Specifies the data type is `'hins'`.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMJapaneseWeightType`
Specifies the data type is short integer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMJapanesePhoneticType`
Specifies the data type is Unicode text.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMJapaneseAccentType`
Specifies the data type is `byte`.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMJapaneseOnKunReadingType`
Specifies the data type is Unicode text.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMJapaneseFukugouInfoType`
Specifies the data type is `'fuku'`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Field Info Record Entries

Specifies tags for the entries in a field information record.

```
enum {
    keyDCMFieldTag = 'ftag',
    keyDCMFieldType = 'ftyp',
    keyDCMMaxRecordSize = 'mrsz',
    keyDCMFieldAttributes = 'fatr',
    keyDCMFieldDefaultData = 'fdef',
    keyDCMFieldName = 'fnam',
    keyDCMFieldFindMethods = 'ffnd'
};
```

Constants

`keyDCMFieldTag`
The data type of the associated data is `typeEnumeration`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`keyDCMFieldType`
The data type of the associated data is `typeEnumeration`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`keyDCMMaxRecordSize`
The data type of the associated data is `typeMagnitude`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`keyDCMFieldAttributes`
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`keyDCMFieldDefaultData`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`keyDCMFieldName`
The data type of the associated data is `typeChar`.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`keyDCMFieldFindMethods`
The data associated with this field is a list (`typeAEList`) of `typeDCMFindMethod` values.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

Field Info Record Types

Specify special types for a field information record.

```
enum {
    typeDCMFieldAttributes = 'fatr',
    typeDCMFindMethod = 'fmth'
};
```

Constants

`typeDCMFieldAttributes`
Specifies a data type for field attributes.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`typeDCMFindMethod`
Specifies a data type for search methods.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

Listing Permissions

Specifies whether to allow or prohibit a dictionary from being listed.

```
enum {
    kDCMAllowListing = 0,
    kDCMProhibitListing = 1
};
```

Constants

`kDCMAllowListing`

Specifies to allow listing.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMProhibitListing`

Specifies to prohibit listing.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Discussion

A dictionary that is prohibited from being listed is available only to the application that created the dictionary.

Permission Levels

Specify permission levels for a dictionary.

```
enum {
    kDCMReadOnlyDictionary = 0,
    kDCMReadWriteDictionary = 1
};
```

Constants

`kDCMReadOnlyDictionary`

Specifies the dictionary is read-only.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMReadWriteDictionary`

Specifies the dictionary has read-write permission.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Search Methods

Specify search criteria.

```
enum {
    kDCMFindMethodExactMatch = kAEEquals,
    kDCMFindMethodBeginningMatch = kAEBeginsWith,
    kDCMFindMethodContainsMatch = kAEContains,
    kDCMFindMethodEndingMatch = kAEEndsWith,
    kDCMFindMethodForwardTrie = 'ftri',
    kDCMFindMethodBackwardTrie = 'btri'
};
typedef OSType DCMFindMethod;
```

Constants

`kDCMFindMethodExactMatch`
Specifies an exact match.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMFindMethodBeginningMatch`
Specifies the beginning must match. For example, cat matches catch and catalog.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMFindMethodContainsMatch`
Specifies the match can be contained in a string. (rat matches crater, decorate)
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMFindMethodEndingMatch`
Specifies the end must match. For example, bat matches combat and acrobat.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMFindMethodForwardTrie`
Specifies partial character string from the front. For example, theme matches the, them, and theme. Used for morphological analysis.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

`kDCMFindMethodBackwardTrie`
Specfies partial character string from the back. For example, flash matches ash, lash, and flash. Used for morphological analysis.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Dictionary.h`.

Wild Card Values

Represent any field tag or field type.

```
enum {
    kDCMAnyFieldTag = typeWildCard,
    kDCMAnyFieldType = typeWildCard
};
```

Constants

`kDCMAnyFieldTag`

Specifies any field.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

`kDCMAnyFieldType`

Specifies any type.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Dictionary.h`.

Result Codes

The most common result codes returned by Dictionary Manager are listed below.

Result Code	Value	Description
`dcmParamErr`	-7100	Bad parameter. Available in Mac OS X v10.0 and later.
`dcmNotDictionaryErr`	-7101	Not a dictionary. Available in Mac OS X v10.0 and later.
`dcmBadDictionaryErr`	-7102	Invalid dictionary. Available in Mac OS X v10.0 and later.
`dcmPermissionErr`	-7103	Invalid permission. Available in Mac OS X v10.0 and later.
`dcmDictionaryNotOpenErr`	-7104	Dictionary not opened. Available in Mac OS X v10.0 and later.
`dcmDictionaryBusyErr`	-7105	Dictionary is busy. Available in Mac OS X v10.0 and later.
`dcmBlockFullErr`	-7107	Dictionary block is full. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
dcmNoRecordErr	-7108	No such record. Available in Mac OS X v10.0 and later.
dcmDupRecordErr	-7109	Same record already exists. Available in Mac OS X v10.0 and later.
dcmNecessaryFieldErr	-7110	Missing the required field. Available in Mac OS X v10.0 and later.
dcmBadFieldInfoErr	-7111	Incomplete field. Available in Mac OS X v10.0 and later.
dcmBadFieldTypeErr	-7112	No such field type supported. Available in Mac OS X v10.0 and later.
dcmNoFieldErr	-7113	No such field exists. Available in Mac OS X v10.0 and later.
dcmBadKeyErr	-7115	Bad key information. Available in Mac OS X v10.0 and later.
dcmTooManyKeyErr	-7116	Too many key fields. Available in Mac OS X v10.0 and later.
dcmBadDataSizeErr	-7117	Data size too large. Available in Mac OS X v10.0 and later.
dcmBadFindMethodErr	-7118	Search method not supported. Available in Mac OS X v10.0 and later.
dcmBadPropertyErr	-7119	No such property exists. Available in Mac OS X v10.0 and later.
dcmProtectedErr	-7121	Need a keyword to use the dictionary. Available in Mac OS X v10.0 and later.
dcmNoAccessMethodErr	-7122	No such access method exists. Available in Mac OS X v10.0 and later.
dcmBadFeatureErr	-7124	Invalid access method feature. Available in Mac OS X v10.0 and later.
dcmIterationCompleteErr	-7126	Iteration complete; no more items in the iterator. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
dcmBufferOverflowErr	-7127	Data is larger than the buffer size. Available in Mac OS X v10.0 and later.

Display Manager Reference (Not Recommended)

Framework: Carbon/Carbon.h

Declared in Displays.h

Overview

Important: The Display Manager is deprecated in Mac OS X version 10.4 and later. The replacement is Quartz Display Services, a modern Mac OS X API that provides similar functionality. For more information, see *Quartz Display Services Reference*.

In Mac OS 9 and earlier, the Display Manager allowed users to dynamically change the arrangement and display modes of the monitors attached to their computers. The Display Manager was included in Carbon to facilitate the porting of legacy applications to Mac OS X. You should not use Display Manager functions in new application development. Instead, you should use Quartz Display Services.

Functions by Task

Adding and Removing Video Devices From the Device List

`DMAddDisplay` (page 1090) Deprecated in Mac OS X v10.4

Adds the `GDevice` structure for a video device to the device list. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMDisposeDisplay` (page 1096) Deprecated in Mac OS X v10.4

Disposes of the `GDevice` structure for a video device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMDisposeList` (page 1097) Deprecated in Mac OS X v10.4

Disposes of a display mode list built by `DMNewDisplayModeList`. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetIndexedDisplayModeFromList` (page 1107) Deprecated in Mac OS X v10.4

Obtains a display mode from the display mode list built by `DMNewDisplayModeList`. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMNewDisplay` (page 1115) Deprecated in Mac OS X v10.4

Adds a video device to the device list and makes the device active. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMNewDisplayModeList` (page 1117) Deprecated in Mac OS X v10.4
Builds a new display mode list for a specified video device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMRemoveDisplay` (page 1119) Deprecated in Mac OS X v10.4
Removes a video device from the device list. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

Changing Display Modes and Display Configurations

`DMBeginConfigureDisplays` (page 1091) Deprecated in Mac OS X v10.4
Allows your application to configure displays. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMBlockMirroring` (page 1092) Deprecated in Mac OS X v10.4
Disables video mirroring. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMDisableDisplay` (page 1095) Deprecated in Mac OS X v10.4
Makes a video device inactive by removing its display area from the desktop. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMEnableDisplay` (page 1099) Deprecated in Mac OS X v10.4
Reactivates a display made inactive with the function `DMDisableDisplay`. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMEndConfigureDisplays` (page 1099) Deprecated in Mac OS X v10.4
Ends configuration begun by `DMBeginConfigureDisplays`. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMMirrorDevices` (page 1111) Deprecated in Mac OS X v10.4
Turns on video mirroring. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMMoveDisplay` (page 1112) Deprecated in Mac OS X v10.4
Moves the boundary rectangle for a video device. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMSetDisplayMode` (page 1125) Deprecated in Mac OS X v10.4
Sets the display mode and pixel depth for a video device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMSetMainDisplay` (page 1126) Deprecated in Mac OS X v10.4
Sets a display to be the main screen. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMUnblockMirroring` (page 1127) Deprecated in Mac OS X v10.4
Reenables video mirroring disabled by the function `DMUnblockMirroring`. You should generally never need to use this function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMUnmirrorDevice` (page 1127) Deprecated in Mac OS X v10.4
Turns off video mirroring. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

Determining Display Modes and Display Configurations

`DMCanMirrorNow` (page 1093) Deprecated in Mac OS X v10.4
Determines whether video mirroring can be activated on the user's computer system. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMCheckDisplayMode` (page 1094) Deprecated in Mac OS X v10.4
Determines if a video device supports a particular display mode and pixel depth. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetAVPowerState` (page 1100) Deprecated in Mac OS X v10.4
Obtains the current power state of a display. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetDisplayMode` (page 1103) Deprecated in Mac OS X v10.4
Obtains the current display mode of a specified video display. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetGraphicInfoByAVID` (page 1106) Deprecated in Mac OS X v10.4
Obtains information about the graphic display of a display device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetNameByAVID` (page 1108) Deprecated in Mac OS X v10.4
Obtains the name of a display device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMIsMirroringOn` (page 1110) Deprecated in Mac OS X v10.4
Determines if video mirroring is active. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMQDIsMirroringCapable` (page 1117) Deprecated in Mac OS X v10.4
Determines if QuickDraw supports video mirroring on the user's system. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMSaveScreenPrefs` (page 1122) Deprecated in Mac OS X v10.4
Saves the user's screen configuration preferences. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMSetAVPowerState` (page 1123) Deprecated in Mac OS X v10.4
Sets the power state of a display device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

Getting Video Devices

`DMGetDisplayIDByGDevice` (page 1102) Deprecated in Mac OS X v10.4
Obtains the display ID number for a video device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetFirstScreenDevice` (page 1104) Deprecated in Mac OS X v10.4
Returns a handle for the first video device in the device list. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetGDeviceByDisplayID` (page 1105) Deprecated in Mac OS X v10.4
Obtains a handle for the video device with a specified display ID. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetNextMirroredDevice` (page 1108) Deprecated in Mac OS X v10.4
Obtains a handle for a video device that mirrors another specified video device. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMGetNextScreenDevice` (page 1109) Deprecated in Mac OS X v10.4
Returns a handle for the next video device in the device list. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

Registering and Unregistering Your Program

`DMRegisterExtendedNotifyProc` (page 1118) Deprecated in Mac OS X v10.4
Registers a function that responds to a Display Notice event outside of an event loop. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMRemoveExtendedNotifyProc` (page 1120) Deprecated in Mac OS X v10.4
Removes your Display Notice event-handling function registered by the `DMRegisterExtendedNotifyProc` function. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DMSendDependentNotification` (page 1122) Deprecated in Mac OS X v10.4
Notifies dependent displays of changes in depth mode or configuration. (Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

Working With Universal Procedure Pointers for Display Manager Callbacks

`DisposeDMComponentListIteratorUPP` (page 1088) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DisposeDMDisplayListIteratorUPP` (page 1088) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DisposeDMDisplayModeListIteratorUPP` (page 1089) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DisposeDMExtendedNotificationUPP` (page 1089) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DisposeDMNotificationUPP` (page 1089) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`DisposeDMProfileListIteratorUPP` (page 1090) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`InvokeDMComponentListIteratorUPP` (page 1128) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`InvokeDMDisplayListIteratorUPP` (page 1129) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`InvokeDMDisplayModeListIteratorUPP` (page 1129) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`InvokeDMExtendedNotificationUPP` (page 1129) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

`InvokeDMNotificationUPP` (page 1130) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

InvokeDMProfileListIteratorUPP (page 1130) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

NewDMComponentListIteratorUPP (page 1130) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

NewDMDisplayListIteratorUPP (page 1131) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

NewDMDisplayModeListIteratorUPP (page 1131) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

NewDMExtendedNotificationUPP (page 1131) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

NewDMNotificationUPP (page 1131) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

NewDMProfileListIteratorUPP (page 1132) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

Miscellaneous

DMConfirmConfiguration (page 1095) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMDisposeAVComponent (page 1096) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMDrawDesktopRect (page 1098) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMDrawDesktopRegion (page 1098) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMGetDeskRegion (page 1101) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMGetDeviceAVIDByPortAVID (page 1101) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMGetDeviceComponentByAVID (page 1102) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMGetDisplayComponent (page 1102) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMGetEnableByAVID (page 1104) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMGetIndexedComponentFromList (page 1106) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMGetPortComponentByAVID (page 1110) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMNewAVDeviceList (page 1113) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMNewAVEngineList (page 1113) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMNewAVIDByDeviceComponent (page 1114) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMNewAVIDByPortComponent (page 1114) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMNewAVPanelList (page 1114) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMNewAVPortListByDeviceAVID (page 1115) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMNewAVPortListByPortType (page 1115) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMRegisterNotifyProc (page 1119) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMRemoveNotifyProc (page 1121) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMResolveDisplayComponents (page 1121) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMSetDisplayComponent (page 1124) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

DMSetEnableByAVID (page 1126) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

Functions

DisposeDMComponentListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DisposeDMComponentListIteratorUPP (
   DMComponentListIteratorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

DisposeDMDisplayListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DisposeDMDisplayListIteratorUPP (
   DMDisplayListIteratorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

DisposeDMDisplayModeListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DisposeDMDisplayModeListIteratorUPP (
   DMDisplayModeListIteratorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

DisposeDMExtendedNotificationUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DisposeDMExtendedNotificationUPP (
   DMExtendedNotificationUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

DisposeDMNotificationUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DisposeDMNotificationUPP (
   DMNotificationUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

DisposeDMProfileListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DisposeDMProfileListIteratorUPP (
   DMProfileListIteratorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

DMAddDisplay

Adds the GDevice structure for a video device to the device list. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMAddDisplay (
   GDHandle newDevice,
   short driver,
   UInt32 mode,
   UInt32 reserved,
   UInt32 displayID,
   Component displayComponent,
   Handle displayState
);
```

Parameters

newDevice
A handle to the GDevice structure for the video device you want to add to the device list. The function DMNewDisplay (page 1115) usually initializes this structure.

driver
The reference number of the graphics device which you are adding to the device list. For most video devices, this information is set at system startup. The function DMAddDisplay passes the number supplied in this parameter to the InitGDevice function in its gdRefNum parameter.

mode
The depth mode. Used by the video device driver, this value sets the pixel depth and specifies color. The function DMAddDisplay passes the value supplied here to the function InitGDevice in its mode parameter.

reserved
Reserved for future expansion. Pass NULL in this parameter.

displayID
A unique identification for the display. For new displays, supply this parameter with the value 0, which causes the Display Manager to generate a unique display ID for this device. If this display was removed, then pass the display ID number of the current display in this parameter.

displayComponent
Reserved for future expansion. Pass NULL in this parameter.

displayState
If your application called DMNewDisplay (page 1115), you must pass the displayState handle obtained. Otherwise pass NULL in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
The DMAddDisplay function adds the display specified by the newDevice parameter as inactive. However, if the specified display is the only display, the Display Manager automatically makes it active. Otherwise, you must call the function DMEnableDisplay (page 1099) to make the specified display active.

The function DMNewDisplay (page 1115) automatically calls DMAddDisplay and DMEnableDisplay. The only time you ned to call DMAddDisplay directly is after the device has been removed by DMRemoveDisplay (page 1119) but not yet disposed of by DMDisposeDisplay (page 1096).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Generally, your application should not use this function, but should instead allow system software to maintain the device list. This function is described here for completeness only.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Displays.h

DMBeginConfigureDisplays

Allows your application to configure displays. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMBeginConfigureDisplays (
   Handle *displayState
);
```

Parameters

displayState
On return, a pointer to a handle to internal Display Manager information about the current display state. The DMEndConfigureDisplays (page 1099) function and many other functions require this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
The `DMBeginConfigureDisplays` function tells the Display Manager to postpone Display Manager configuration checking, the rebuilding of desktop regions, and Apple event notification of Display Manager changes until your application uses the `DMEndConfigureDisplays` function.

You should call the function `DMBeginConfigureDisplays` before calling other Display Manager functions that configure the user's display. When calling functions that configure displays, you should pass the handle obtained by the `DMBeginConfigureDisplays` function. `DMBeginConfigureDisplays` causes system software to wait for your application to complete display changes before managing additional Display Manager events. When your application completes configuring the display environment, call the function `DMEndConfigureDisplays`.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMBlockMirroring

Disables video mirroring. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMBlockMirroring (
   void
);
```

Parameters

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
The function `DMBlockMirroring` disables video mirroring until the user restarts the computer or until an application calls the function `DMUnblockMirroring` (page 1127).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMCanMirrorNow

Determines whether video mirroring can be activated on the user's computer system. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMCanMirrorNow (
   Boolean *canMirrorNow
);
```

Parameters

canMirrorNow
A pointer to a Boolean value; true indicates that mirroring can be activated; false indicates it cannot.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
In the value pointed to by the `canMirrorNow` parameter, the `DMCanMirrorNow` function reports whether video mirroring can be activated. When the `canMirrorNow` parameter points to a value of true, then the computer uses a version of QuickDraw that supports video mirroring, has exactly two displays attached, and does not have mirror blocking in effect.

You can use the `DMQDIsMirroringCapable` (page 1117) function to determine whether the computer uses a version of QuickDraw that supports video mirroring. You can use the `DMBlockMirroring` (page 1092) function and the `DMUnblockMirroring` (page 1127) function to block and unblock video mirroring. To determine whether the user's computer system currently uses video mirroring, use the `DMIsMirroringOn` (page 1110) function.

Special Considerations

The `DMCanMirrorNow` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Version Notes
As of System Software version 7.5, only PowerBook computers support video mirroring.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMCheckDisplayMode

Determines if a video device supports a particular display mode and pixel depth. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMCheckDisplayMode (
   GDHandle theDevice,
   UInt32 mode,
   UInt32 depthMode,
   UInt32 *switchFlags,
   UInt32 reserved,
   Boolean *modeOk
);
```

Parameters

theDevice
A handle to the `GDevice` structure for the video device whose display mode and pixel depth you wish to check.

mode
The display mode you wish to check. You get a list of display modes by calling `DMGetDisplayMode` (page 1103).

depthMode
The pixel depth you wish to check. See "Video Depth Mode Values" for list of possible values.

switchFlags
On return, a pointer to a long integer that indicates if a video device will support the mode specified by the `mode` parameter and the pixel depth specified by the `depthMode` parameter. See "Switch Flags" (page 1163) for a description.

reserved
Reserved for future expansion. Pass `NULL` in this parameter.

modeOk
On return, a pointer to a `Boolean`. If `modeOk` points to a value of `true`, the user or your application can switch the display mode for the video device to the one specified by `mode`.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
Usually, your application only needs to know if a video device supports a specific pixel depth. Thus your application can use the Color QuickDraw function `HasDepth`. The function `DMCheckDisplayMode` is essentially obsolete, and is here for completeness.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMConfirmConfiguration

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMConfirmConfiguration (
   DMModalFilterUPP filterProc,
   UInt32 confirmFlags,
   UInt32 reserved,
   Handle displayState
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMDisableDisplay

Makes a video device inactive by removing its display area from the desktop. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMDisableDisplay (
   GDHandle disableDevice,
   Handle displayState
);
```

Parameters

`disableDevice`
A handle to the `GDevice` structure for the video device whose display you wish to disable.

`displayState`
If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
You are not allowed to disable the last remaining display. Doing so will simply re-enable it. If you want to remove the last remaining display, thereby enabling the `GDevice` structure not associated with any video device, call the function `DMRemoveDisplay` (page 1119).

If you specify the device for the main screen in the `disableDevice` parameter, then `DMDisableDisplay` picks another device and makes it the new main screen.

If `DMDisableDisplay` results in setting a new main screen, the handle you pass in the `disableDevice` parameter does not point to the same `GDevice` structure after `DMDisableDisplay` completes; instead, it points to the `GDevice` structure for the new main screen. If you need to recover the `GDevice` structure for the device you disabled, determine its display ID by using the function `DMGetDisplayIDByGDevice` (page 1102) before calling `DMDisableDisplay`. Then use the function `DMGetGDeviceByDisplayID` (page 1105) to obtain its structure.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMDisposeAVComponent

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMDisposeAVComponent (
   Component theAVComponent
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMDisposeDisplay

Disposes of the `GDevice` structure for a video device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMDisposeDisplay (
   GDHandle disposeDevice,
   Handle displayState
);
```

Parameters

disposeDevice

A handle to the `GDevice` structure for a video device you want to delete.

displayState

If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Discussion

The `DMDisposeDisplay` function disposes of a `GDevice` structure, releases the space allocated for it, and disposes of all the data structures allocated for it. The Display Manager calls this function when appropriate.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Generally, your application should not use this function, but should instead allow system software to maintain the device list. This function is described here for completeness only.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMDisposeList

Disposes of a display mode list built by `DMNewDisplayModeList`. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMDisposeList (
   DMListType panelList
);
```

Parameters

panelList

A value that specifies the display mode list you want to delete.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Discussion

You should call the `DMDisposeList` function after you have iterated the mode list.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Generally, your application should not use this function, but should instead allow system software to maintain the device list. This function is described here for completeness only.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMDrawDesktopRect

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DMDrawDesktopRect (
   Rect *globalRect
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMDrawDesktopRegion

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void DMDrawDesktopRegion (
   RgnHandle globalRgn
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMEnableDisplay

Reactivates a display made inactive with the function `DMDisableDisplay`. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMEnableDisplay (
   GDHandle enableDevice,
   Handle displayState
);
```

Parameters

enableDevice

A handle to the `GDevice` structure for the video device whose display you wish to make active.

displayState

If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Discussion

The function `DMEnableDisplay` reactivates the specified video device by adding its display area to the desktop.

If you add a display with the function `DMAddDisplay` (page 1090) and there are no active displays, the Display Manager will enable the added display.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMEndConfigureDisplays

Ends configuration begun by `DMBeginConfigureDisplays`. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMEndConfigureDisplays (
   Handle displayState
);
```

Parameters

displayState

Supply this parameter with the handle obtained by the `DMBeginConfigureDisplays` (page 1091) function.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Discussion

The function `DMEndConfigureDisplays` resumes Display Manager configuration checking, the rebuilding of desktop regions, and Apple event notification of Display Manager changes, all of which are postponed when you use the function `DMBeginConfigureDisplays` (page 1091). Your application will then receive a single Display Notice event notifying your application of Display Manager changes, and your application can manage its windows accordingly.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMGetAVPowerState

Obtains the current power state of a display. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetAVPowerState (
   AVIDType theID,
   AVPowerStatePtr getPowerState,
   UInt32 reserved1
);
```

Parameters

theID

The ID number of the display device whose power state you want to obtain.

getPowerState
A pointer to a structure of type `AVPowerStateRec` (page 1137). On return, this parameter points to a value specifying the current power state of display device.

reserved1
Reserved for future expansion. Pass NULL in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetDeskRegion

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetDeskRegion (
   RgnHandle *desktopRegion
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetDeviceAVIDByPortAVID

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetDeviceAVIDByPortAVID (
   AVIDType portAVID,
   AVIDType *deviceAVID
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetDeviceComponentByAVID

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetDeviceComponentByAVID (
   AVIDType theDeviceID,
   Component *theDeviceComponent,
   ComponentDescription *theDesciption,
   ResType *theDeviceKind
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetDisplayComponent

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetDisplayComponent (
   GDHandle theDevice,
   Component *displayComponent
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetDisplayIDByGDevice

Obtains the display ID number for a video device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetDisplayIDByGDevice (
   GDHandle displayDevice,
   DisplayIDType *displayID,
   Boolean failToMain
);
```

Parameters

displayDevice

A handle to the `GDevice` structure for the video device whose display ID you wish to obtain.

displayID

On return, a pointer to the display ID for the video device specified by the `displayDevice` parameter.

failToMain

If `true` and the specified video device does not have a display ID, on return the function sets the `displayID` parameter to a pointer to the display ID of the video device for the main screen. If `false` and the specified video device does not have a display ID, the function returns the `kDMDisplayNotFoundErr` result code.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMGetDisplayMode

Obtains the current display mode of a specified video display. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetDisplayMode (
   GDHandle theDevice,
   VDSwitchInfoPtr switchInfo
);
```

Parameters

theDevice

A handle to the `GDevice` structure for the video device whose display mode you wish to obtain.

switchInfo

On return, a pointer to an internal Display Manager structure containing display mode information.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMGetEnableByAVID

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetEnableByAVID (
   AVIDType theAVID,
   Boolean *isAVIDEnabledNow,
   Boolean *canChangeEnableNow
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMGetFirstScreenDevice

Returns a handle for the first video device in the device list. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
GDHandle DMGetFirstScreenDevice (
   Boolean activeOnly
);
```

Parameters

`activeOnly`

If `true`, the `DMGetFirstScreenDevice` function returns a handle to the first of all active video devices. If `false`, the function returns a handle to the first of all video devices, active or not. You may use the Active Device Constants in this parameter. See "Active Device Only Values" (page 1148).

Return Value

If `activeOnly` is `true`, a handle to the `GDevice` structure for the first active video device. If `activeOnly` is `false`, a handle to the `GDevice` structure for the first video device. See the QuickDraw Manager documentation for a description of the `GDHandle` data type.

Discussion

The `DMGetFirstScreenDevice` function is useful if you want to find out more about the current mode.

You can use the function `DMGetNextScreenDevice` (page 1109) to loop through all of the video devices in the device list.

The `DMGetFirstScreenDevice` function is similar to the QuickDraw function `GetDeviceList`, except that when returning `GDevice` structures, `GetDeviceList` does not distinguish between inactive and active video devices or between the `GDevice` structures for video devices and the `GDevice` structures associated with no video devices.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetGDeviceByDisplayID

Obtains a handle for the video device with a specified display ID. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetGDeviceByDisplayID (
   DisplayIDType displayID,
   GDHandle *displayDevice,
   Boolean failToMain
);
```

Parameters

displayID
The display ID for the video device whose handle you wish to obtain.

displayDevice
On return, a pointer to the handle to the `GDevice` structure for the video device specified by the `displayID` parameter.

failToMain
If `true` and there is no video device associated with the `displayID` parameter, on return the function sets `displayDevice` to a pointer to the handle for the video device for the main screen. If `false` and there is no video device associated with the `displayID` parameter, the function returns the `kDMDisplayNotFoundErr` result code.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Displays.h`

DMGetGraphicInfoByAVID

Obtains information about the graphic display of a display device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetGraphicInfoByAVID (
   AVIDType theID,
   PicHandle *theAVPcit,
   Handle *theAVIconSuite,
   AVLocationRec *theAVLocation
);
```

Parameters

theID
The ID number of the display device whose information you want to obtain.

theAVPcit
On return, a pointer to the handle for the picture structure you want to get.

theAVIconSuite
On return, a pointer to a handle whose structure reports the icon suite for a display device.

theAVLocation
On return, a pointer to the location structure for the device you want information about.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetIndexedComponentFromList

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetIndexedComponentFromList (
   DMListType panelList,
   DMListIndexType itemIndex,
   UInt32 reserved,
   DMComponentListIteratorUPP listIterator,
   void *userData
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetIndexedDisplayModeFromList

Obtains a display mode from the display mode list built by `DMNewDisplayModeList`. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetIndexedDisplayModeFromList (
   DMListType panelList,
   DMListIndexType itemIndex,
   UInt32 reserved,
   DMDisplayModeListIteratorUPP listIterator,
   void *userData
);
```

Parameters

panelList
A value that specifies the list from which to obtain information about the display modes created by the function `DMNewDisplayModeList` (page 1117).

itemIndex
A value that specifies the index of the display mode you wish to obtain.

reserved
Reserved for future expansion. Pass `NULL` in this parameter.

listIterator
A universal procedure pointer. The iterator this pointer specifies supplies the function to be called with the information about the display mode specified by `theListCount`.

userData
A pointer you pass for `listIterator` usually used to obtain information about the display mode from the UPP and return it to the caller of `DMGetIndexedDisplayModeFromList`.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Generally, your application should not use this function, but should instead allow system software to maintain the device list. This function is described here for completeness only.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetNameByAVID

Obtains the name of a display device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetNameByAVID (
   AVIDType theID,
   UInt32 nameFlags,
   Str255 name
);
```

Parameters

theID
The ID number of the display device whose name you want to obtain.

nameFlags
Reserved for future expansion. Pass `NULL` in this parameter.

name
On return, a string containing the name of the display device specified by the parameter `theID`.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
An AVID is really a display ID as an AVID references a video display just like a display ID. Developers planned to use AVIDs for an extended set of devices, however, they never did this.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetNextMirroredDevice

Obtains a handle for a video device that mirrors another specified video device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetNextMirroredDevice (
   GDHandle gDevice,
   GDHandle *mirroredDevice
);
```

Parameters

gDevice

A handle to the `GDevice` structure for the video device that another video device mirrors.

mirroredDevice

On return, a pointer to the handle for the video device that displays a mirror image of the device specified in the `gDevice` parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMGetNextScreenDevice

Returns a handle for the next video device in the device list. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
GDHandle DMGetNextScreenDevice (
   GDHandle theDevice,
   Boolean activeOnly
);
```

Parameters

theDevice

A handle to the `GDevice` structure at which you want the function to begin. You can supply the handle returned by the function `DMGetFirstScreenDevice` or `DMGetNextScreenDevice`.

activeOnly

If `true`, the `DMGetNextScreenDevice` function returns a handle for the next active video device. If `false`, `DMGetNextScreenDevice` returns a handle for the next video device, active or not. You may use the Active Device Constants in this parameter. See "Active Device Only Values" (page 1148).

Return Value

If `activeOnly` is `true`, a handle to the next `GDevice` structure for an active video device. If `activeOnly` is `false`, a handle to the next `GDevice` structure for a video device. If there are no more `GDevice` structures in the list, `DMGetNextScreenDevice` returns `NULL`. See the QuickDraw Manager documentation for a description of the `GDHandle` data type.

Discussion
The `DMGetNextScreenDevice` function is similar to the QuickDraw function `GetNextDevice`, except that when returning `GDevice` structures, `GetNextDevice` does not distinguish between inactive and active video devices or between the `GDevice` structures for video devices and the `GDevice` structures associated with no video devices.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMGetPortComponentByAVID

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMGetPortComponentByAVID (
   DisplayIDType thePortID,
   Component *thePortComponent,
   ComponentDescription *theDesciption,
   ResType *thePortKind
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMIsMirroringOn

Determines if video mirroring is active. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMIsMirroringOn (
   Boolean *isMirroringOn
);
```

Parameters

`isMirroringOn`
On return, a pointer to a Boolean value; `true` indicates that mirroring is on; `false` indicates it is not.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMMirrorDevices

Turns on video mirroring. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMMirrorDevices (
   GDHandle gD1,
   GDHandle gD2,
   Handle displayState
);
```

Parameters

gD1

A handle to the `GDevice` structure for the video device whose pixel image you want duplicated on another device.

gD2

A handle to the `GDevice` structure for the video device on which you want to duplicate the pixel image specified in the `gD1` parameter.

displayState

If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Discussion

Your application should leave control of video mirroring to the user. However, if video mirroring is useful for your application (for example, if your application displays on-screen presentations), you might provide a control so that the user can switch to video mirroring directly from your application. In this case, `DMMirrorDevices` is useful to your application. Your control should also allow the user to turn video mirroring off; the function `DMUnmirrorDevice` (page 1127) supports this.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Displays.h`

DMMoveDisplay

Moves the boundary rectangle for a video device. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMMoveDisplay (
   GDHandle moveDevice,
   short x,
   short y,
   Handle displayState
);
```

Parameters

moveDevice
A handle to the `GDevice` structure for the video device whose boundary rectangle you wish to move.

x
The horizontal coordinate on the QuickDraw global coordinate plane for the point to which you want to move the upper-left corner of the boundary rectangle.

y
The vertical coordinate on the QuickDraw global coordinate plane for the point to which you want to move the upper-left corner of the boundary rectangle.

displayState
If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
The `DMMoveDisplay` function moves the boundary rectangle for the specified video device to the point (`x` , `y`) in the QuickDraw global coordinate plane. If the video device controls the main screen, which always has the global coordinates (`0` , `0`), then all other video devices are offset by horizontal distance `x` and vertical distance `y`.

A boundary rectangle is the rectangle that links the local coordinate system of a graphics port to QuickDraw's global coordinate system and defines the area of the pixel image or bit image into which QuickDraw can draw. The boundary rectangle is stored in either the pixel map or the bitmap contained in a `GDevice` structure.

The Display Manager will reposition overlapped or discontiguous boundary rects to create a non-overlapping contiguous desktop space.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application

should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMNewAVDeviceList

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewAVDeviceList (
   ResType deviceType,
   UInt32 deviceListFlags,
   UInt32 reserved,
   DMListIndexType *deviceCount,
   DMListType *deviceList
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMNewAVEngineList

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewAVEngineList (
   DisplayIDType displayID,
   ResType engineType,
   DMFidelityType minimumFidelity,
   UInt32 engineListFlags,
   UInt32 reserved,
   DMListIndexType *engineCount,
   DMListType *engineList
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMNewAVIDByDeviceComponent

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewAVIDByDeviceComponent (
   Component theDeviceComponent,
   ResType portKind,
   UInt32 reserved,
   DisplayIDType *newID
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMNewAVIDByPortComponent

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewAVIDByPortComponent (
   Component thePortComponent,
   ResType portKind,
   UInt32 reserved,
   AVIDType *newID
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMNewAVPanelList

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewAVPanelList (
   DisplayIDType displayID,
   ResType panelType,
   DMFidelityType minimumFidelity,
   UInt32 panelListFlags,
   UInt32 reserved,
   DMListIndexType *thePanelCount,
   DMListType *thePanelList
);
```

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Displays.h

DMNewAVPortListByDeviceAVID

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewAVPortListByDeviceAVID (
   AVIDType theID,
   DMFidelityType minimumFidelity,
   UInt32 portListFlags,
   UInt32 reserved,
   DMListIndexType *devicePortCount,
   DMListType *theDevicePortList
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Displays.h

DMNewAVPortListByPortType

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewAVPortListByPortType (
   ResType subType,
   UInt32 portListFlags,
   UInt32 reserved,
   DMListIndexType *devicePortCount,
   DMListType *theDevicePortList
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Displays.h

DMNewDisplay

Adds a video device to the device list and makes the device active. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewDisplay (
   GDHandle *newDevice,
   short driverRefNum,
   UInt32 mode,
   UInt32 reserved,
   DisplayIDType displayID,
   Component displayComponent,
   Handle displayState
);
```

Parameters

newDevice

A pointer to a handle to a `GDevice` structure for the video device that you want to add to the device list.

driverRefNum

The reference number of the video device which you are adding to the device list. This information is usually set at system startup. The function `DMAddDisplay` (page 1090) passes the value supplied here to the `InitGDevice` function in its `gdRefNum` parameter.

mode

The depth mode. Used by the video device driver, this value sets the pixel depth and specifies color. The function `DMAddDisplay` (page 1090) passes the value supplied here to the function `InitGDevice` in its `mode` parameter.

reserved

Reserved for future expansion. Pass `NULL` in this parameter.

displayID

A unique identification for the display. For new displays, supply this parameter with the value 0, which causes the Display Manager to generate a unique display ID for this device. If this display was removed, then pass the display ID of the current display in this parameter.

displayComponent

Reserved for future expansion. Pass `NULL` in this parameter.

displayState

If your application called `DMAddDisplay` (page 1090), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Generally, your application should not use this function, but should instead allow system software to maintain the device list. This function is described here for completeness only.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMNewDisplayModeList

Builds a new display mode list for a specified video device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMNewDisplayModeList (
   DisplayIDType displayID,
   UInt32 modeListFlags,
   UInt32 reserved,
   DMListIndexType *thePanelCount,
   DMListType *thePanelList
);
```

Parameters

displayID
The display ID for the video device that will have a new display mode list.

modeListFlags
Reserved for future expansion. Pass `NULL` in this parameter.

reserved
Reserved for future expansion. Pass `NULL` in this parameter.

thePanelCount
The number of entries in the display mode list specified by the `theList` parameter.

thePanelList
The display mode list for the specified video device.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Generally, your application should not use this function, but should instead allow system software to maintain the device list. This function is described here for completeness only.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In

`Displays.h`

DMQDIsMirroringCapable

Determines if QuickDraw supports video mirroring on the user's system. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMQDIsMirroringCapable (
   Boolean *qdIsMirroringCapable
);
```

Parameters

qdIsMirroringCapable

On return, a pointer to the value `true` if QuickDraw supports video mirroring; otherwise, a pointer to the value `false`.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Displays.h`

DMRegisterExtendedNotifyProc

Registers a function that responds to a Display Notice event outside of an event loop. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMRegisterExtendedNotifyProc (
   DMExtendedNotificationUPP notifyProc,
   void *notifyUserData,
   unsigned short nofifyOnFlags,
   DMProcessInfoPtr whichPSN
);
```

Parameters

notifyProc

A pointer to your function that handles a Display Notice event.

notifyUserData

A pointer to caller-specific information which the Display Manager will return to your application when you request it.

notifyOnFlags

Reserved for future expansion. You should pass `kNilOptions` in this parameter.

whichPSN

A pointer to the Process Serial Number associated with your Display Notice event-handling function. If this process terminates, the Display Notice event-handling function is automatically removed. For example, the Monitors control panel supplies the Finder's process number when registering its Display Notice event-handling function.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
When the Display Manager sends your function the Display Notice event, your application or utility should respond by moving or resizing its windows and updating any internally-maintained video device information as appropriate.

When you are finished with your notification function, remove it by calling `DMRemoveExtendedNotifyProc` (page 1120).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Displays.h`

DMRegisterNotifyProc

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMRegisterNotifyProc (
   DMNotificationUPP notificationProc,
   DMProcessInfoPtr whichPSN
);
```

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Displays.h`

DMRemoveDisplay

Removes a video device from the device list. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMRemoveDisplay (
   GDHandle removeDevice,
   Handle displayState
);
```

Parameters

removeDevice

A handle to the `GDevice` structure for the video device you want to remove from the device list. The function `DMRemoveDisplay` does not actually dispose of this structure, but instead removes it from the device list.

displayState
If your application called DMBeginConfigureDisplays (page 1091), you must pass the displayState handle obtained. Otherwise pass NULL in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
The function DMRemoveDisplay may call the function DMSetMainDisplay (page 1126), which causes the removeDevice parameter to contain a handle to the GDevice structure for the new main screen, not the video device whose handle was passed to DMRemoveDisplay. To recover the GDevice structure for the disabled device, determine its display ID by using the function DMGetDisplayIDByGDevice (page 1102) before calling DMRemoveDisplay. Then use the function DMGetGDeviceByDisplayID (page 1105) to obtain the GDevice structure for the specified device.

You are not allowed to disable the last remaining display using the DMDisableDisplay (page 1095) function. Doing so will simply re-enable it. If you want to remove the last remaining display, thereby enabling the GDevice structure not associated with any video device, you must call DMRemoveDisplay.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Generally, your application should not use this function, but should instead allow system software to maintain the device list. This function is described here for completeness only.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Displays.h

DMRemoveExtendedNotifyProc

Removes your Display Notice event-handling function registered by the DMRegisterExtendedNotifyProc function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMRemoveExtendedNotifyProc (
   DMExtendedNotificationUPP notifyProc,
   void *notifyUserData,
   DMProcessInfoPtr whichPSN,
   unsigned short removeFlags
);
```

Parameters

notifyProc
A pointer to your function you want to remove that handles a Display Notice event.

notifyUserData
A pointer to caller-specific information which the Display Manager will return to your application when you request it.

whichPSN

A pointer to the Process Serial Number associated with your Display Notice event-handling function. If this process terminates, the Display Notice event-handling function is automatically removed. For example, the Monitors control panel supplies the Finder's process number when registering its Display Notice event-handling function.

removeFlags

Reserved for future expansion. You should pass `kNilOptions` in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Displays.h`

DMRemoveNotifyProc

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMRemoveNotifyProc (
   DMNotificationUPP notificationProc,
   DMProcessInfoPtr whichPSN
);
```

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Displays.h`

DMResolveDisplayComponents

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMResolveDisplayComponents (
   void
);
```

Parameters

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMSaveScreenPrefs

Saves the user's screen configuration preferences. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMSaveScreenPrefs (
   UInt32 reserved1,
   UInt32 saveFlags,
   UInt32 reserved2
);
```

Parameters

reserved1
Reserved for future expansion. Pass `NULL` in this parameter.

saveFlags
Reserved for future expansion. Pass `NULL` in this parameter.

reserved2
Reserved for future expansion. Pass `NULL` in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
Usually when you change screen properties such as pixel depth, the changes will only be temporary and will usually reset after restarting. However, the function `DMSaveScreenPrefs` makes the current screen properties permanent.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMSendDependentNotification

Notifies dependent displays of changes in depth mode or configuration. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMSendDependentNotification (
   ResType notifyType,
   ResType notifyClass,
   AVIDType displayID,
   ComponentInstance notifyComponent
);
```

Parameters

notifyType

The resource type that identifies the engine that made the change. Examples might be component engines that control brightness, contrast, or screen size. You may pass zero in this parameter. See `DependentNotifyRec` (page 1138) for more information.

notifyClass

The resource type that identifies the class of change the user or engine has made, such as color depth, pixel size, or screen size. See `DependentNotifyRec` (page 1138) for more information.

displayID

The ID number of the dependent display which you want to notify of Display Manager events. On return, the Display Manager sets the `notifyPortID` constant of the `DependentNotifyRec` (page 1138) structure. See DependentNotifyRec (page 1138) for more information.

notifyComponent

A value that notifies the display component what engine, if any, caused a change in a dependent display. You may pass 0 in this parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Discussion

The Display Manager uses the `DMSendDependentNotification` function to send notifications to registered Display Notice event-handling functions. This function uses all its parameters to supply values for the `DependentNotifyRec` (page 1138) structure which is sent out to registrants. Generally, your application does not need to use this function.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMSetAVPowerState

Sets the power state of a display device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMSetAVPowerState (
   AVIDType theID,
   AVPowerStatePtr setPowerState,
   UInt32 powerFlags,
   Handle displayState
);
```

Parameters

theID

The ID number of the display device whose power state you want to change.

setPowerState

On return, this parameter points to a value that your application can use to set the power state of a display device.

powerFlags

A value that specifies the power state to which a display device can be set.

displayState

A handle to internal Display Manager information about the current display state.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMSetDisplayComponent

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMSetDisplayComponent (
   GDHandle theDevice,
   Component displayComponent
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMSetDisplayMode

Sets the display mode and pixel depth for a video device. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMSetDisplayMode (
   GDHandle theDevice,
   UInt32 mode,
   UInt32 *depthMode,
   long reserved,
   Handle displayState
);
```

Parameters

theDevice

A handle to the `GDevice` structure for the video device whose display mode and pixel depth you wish to set.

mode

The number used by a video device to identify its display mode. If you supply the value 0 in this parameter, `DMSetDisplayMode` uses the current display mode. To specify another display mode, use the function `DMNewDisplayModeList` (page 1117).

depthMode

A pointer to the desired pixel depth for the video device specified by `theDevice`. If you pass a pointer to 0, `DMSetDisplayMode` attempts to keep the current depth. If you pass a pointer to 1, 2, 4, 8, 16, or 32, `DMSetDisplayMode` attempts to set the device to use your specified pixel depth. If you supply a pointer to a value of 128 or greater, then `DMSetDisplayMode` sets the depth to the depth mode represented by the Video Depth Mode values. See "Video Depth Mode Values" for more information.

On return, this parameter contains a pointer to the new pixel depth. This value represents the depth mode closest to the one you requested when calling `DMSetDisplayMode`.

reserved

Reserved for future expansion. Pass `NULL` in this parameter.

displayState

If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

DMSetEnableByAVID

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMSetEnableByAVID (
   AVIDType theAVID,
   Boolean doEnable,
   Handle displayState
);
```

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Displays.h`

DMSetMainDisplay

Sets a display to be the main screen. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMSetMainDisplay (
   GDHandle newMainDevice,
   Handle displayState
);
```

Parameters
`newMainDevice`

A handle to the `GDevice` structure for the video device whose display you wish to make the main screen.

`displayState`

If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Discussion
After a call to the function `DMSetMainDisplay`, the handle specified by the parameter `newMainDevice` will point to the `GDevice` structure for the video device whose display, before calling `DMSetMainDisplay`, was the main screen. To obtain a handle to the main screen, you can use the Color QuickDraw function `GetMainDevice`.

`DMSetMainDisplay` moves the menu bar to the display for the video device specified by `newMainDevice`. QuickDraw maps the (0,0) origin point of the global coordinate system to the main screen's upper-left corner, and other screens are positioned adjacent to it.

Special Considerations
Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMUnblockMirroring

Reenables video mirroring disabled by the function `DMUnblockMirroring`. You should generally never need to use this function. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMUnblockMirroring (
   void
);
```

Parameters

Return Value
A result code. See "Display Manager Result Codes" (page 1164).

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Applications generally never need to use this function. In case you find a compelling need to change the user's display configuration, this function is described here for completeness. Note that if your application uses Display Manager functions to change the display configuration of the user's video devices, your application should make these changes only with the consent of the user. If your application must have a specific pixel depth, for example, it should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Displays.h`

DMUnmirrorDevice

Turns off video mirroring. (Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
OSErr DMUnmirrorDevice (
   GDHandle gDevice,
   Handle displayState
);
```

Parameters

gDevice

A handle to the `GDevice` structure for the video device on which you no longer wish to mirror the pixel image of another device.

displayState

If your application called `DMBeginConfigureDisplays` (page 1091), you must pass the `displayState` handle obtained. Otherwise pass `NULL` in this parameter.

Return Value

A result code. See "Display Manager Result Codes" (page 1164).

Discussion

When the function `DMUnmirrorDevice` completes, the display controlled by the video device specified in the `gDevice` parameter no longer contains the mirror image of another display.

Your application should leave control of video mirroring to the user. However, if video mirroring is useful for your application (for example, if your application displays on-screen presentations), you might provide a control so that the user can switch to video mirroring directly from your application. In this case, the function `DMMirrorDevices` (page 1111) is useful for switching video mirroring on, and `DMUnmirrorDevice` function is useful for switching it off again.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Displays.h`

InvokeDMComponentListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void InvokeDMComponentListIteratorUPP (
   void *userData,
   DMListIndexType itemIndex,
   DMComponentListEntryPtr componentInfo,
   DMComponentListIteratorUPP userUPP
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
Displays.h

InvokeDMDisplayListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void InvokeDMDisplayListIteratorUPP (
   void *userData,
   DMListIndexType itemIndex,
   DisplayListEntryPtr displaymodeInfo,
   DMDisplayListIteratorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

InvokeDMDisplayModeListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void InvokeDMDisplayModeListIteratorUPP (
   void *userData,
   DMListIndexType itemIndex,
   DMDisplayModeListEntryPtr displaymodeInfo,
   DMDisplayModeListIteratorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

InvokeDMExtendedNotificationUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void InvokeDMExtendedNotificationUPP (
   void *userData,
   short theMessage,
   void *notifyData,
   DMExtendedNotificationUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

InvokeDMNotificationUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void InvokeDMNotificationUPP (
   AppleEvent *theEvent,
   DMNotificationUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

InvokeDMProfileListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
void InvokeDMProfileListIteratorUPP (
   void *userData,
   DMListIndexType itemIndex,
   DMProfileListEntryPtr profileInfo,
   DMProfileListIteratorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

NewDMComponentListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
DMComponentListIteratorUPP NewDMComponentListIteratorUPP (
   DMComponentListIteratorProcPtr userRoutine
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
Displays.h

NewDMDisplayListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
DMDisplayListIteratorUPP NewDMDisplayListIteratorUPP (
   DMDisplayListIteratorProcPtr userRoutine
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

NewDMDisplayModeListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
DMDisplayModeListIteratorUPP NewDMDisplayModeListIteratorUPP (
   DMDisplayModeListIteratorProcPtr userRoutine
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

NewDMExtendedNotificationUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
DMExtendedNotificationUPP NewDMExtendedNotificationUPP (
   DMExtendedNotificationProcPtr userRoutine
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

NewDMNotificationUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
DMNotificationUPP NewDMNotificationUPP (
   DMNotificationProcPtr userRoutine
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

NewDMProfileListIteratorUPP

(Deprecated in Mac OS X v10.4. Use Quartz Display Services instead; see *Quartz Display Services Reference*.)

```
DMProfileListIteratorUPP NewDMProfileListIteratorUPP (
   DMProfileListIteratorProcPtr userRoutine
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Displays.h`

Callbacks

DMComponentListIteratorProcPtr

```
typedef void (*DMComponentListIteratorProcPtr)
(
    void * userData,
    DMListIndexType itemIndex,
    DMComponentListEntryPtr componentInfo
);
```

If you name your function `MyDMComponentListIteratorProc`, you would declare it like this:

```
void MyDMComponentListIteratorProc (
    void * userData,
    DMListIndexType itemIndex,
    DMComponentListEntryPtr componentInfo
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMDisplayListIteratorProcPtr

```
typedef void (*DMDisplayListIteratorProcPtr)
(
    void * userData,
    DMListIndexType itemIndex,
    DisplayListEntryPtr displaymodeInfo
);
```

If you name your function `MyDMDisplayListIteratorProc`, you would declare it like this:

```
void MyDMDisplayListIteratorProc (
    void * userData,
    DMListIndexType itemIndex,
    DisplayListEntryPtr displaymodeInfo
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMDisplayModeListIteratorProcPtr

Defines a pointer to a list iterator callback function.

```
typedef void (*DMDisplayModeListIteratorProcPtr)
(
    void * userData,
    DMListIndexType itemIndex,
    DMDisplayModeListEntryPtr displaymodeInfo
);
```

If you name your function `MyDMDisplayModeListIteratorProc`, you would declare it like this:

```
void MyDMDisplayModeListIteratorProc (
    void * userData,
    DMListIndexType itemIndex,
    DMDisplayModeListEntryPtr displaymodeInfo
);
```

Parameters

userData
: A pointer to data about mode changes provided by the user.

itemIndex
: Specifies the list entry. See `DMListIndexType` (page 1145) for more information. This is the index passed into `DMGetIndexedDisplayModeFromList` (page 1107).

displaymodeInfo
: A pointer to a structure of type `DMDisplayModeListEntryRec` (page 1143) that provides display mode information.

Discussion

The function `DMGetIndexedDisplayModeFromList` (page 1107) uses this callback function to retrieve and return information about a display mode to the caller of `DMGetIndexedDisplayModeFromList`.

When you implement this function, the pointer you pass to the `DMGetIndexedDisplayModeFromList` function should be a universal procedure pointer with the following type definition:

```
typedef (DMDisplayModeListIteratorProcPtr)
DMDisplayModeListIteratorUPP;
```

To create a universal procedure pointer for your application-defined function, you should use the `NewDMDisplayModeListIteratorUPP` function as follows:

```
DMDisplayModeListIteratorUPP MyDMDisplayModeListIteratorUPP;
MyDMDisplayModeListIteratorUPP = NewDMDisplayModeListIteratorUPP
(&MyDMDisplayModeListIteratorCallback)
```

You can then pass `MyDMDisplayModeListIteratorUPP` in the `listIterator` parameter of the `DMGetIndexedDisplayModeFromList` (page 1107) function. When you no longer need the list iterator, you should dispose of the UPP using the `DisposeDMDisplayModeListIteratorUPP` function:

```
DisposeDMDisplayModeListIteratorUPP (
    MyDMDisplayModeListIteratorUPP);
```

Using this call ensures that the call is made through a universal procedure pointer.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMExtendedNotificationProcPtr

Defines a pointer to an extended notification callback function.

```
typedef void (*DMExtendedNotificationProcPtr)
(
    void * userData,
    short theMessage,
    void * notifyData
);
```

If you name your function `MyDMExtendedNotificationProc`, you would declare it like this:

```
void MyDMExtendedNotificationProc (
    void * userData,
    short theMessage,
    void * notifyData
);
```

Parameters

userData

A pointer you passed into `DMRegisterExtendedNotifyProc` (page 1118).

theMessage

A message selector. See "Notification Messages" (page 1159) for information on specific message selectors.

notifyData

A pointer to message-specific information data provided by the the Display Manager, described in "Notification Messages" (page 1159).

Discussion

Display Manager notification functions use this callback function when your application needs to know when certain events have occurred. The system software may implement these events or follow a user action. When these events occur, the Display Manager will send notification messages to registrants.

When you call the function `DMRegisterExtendedNotifyProc` (page 1118) you designate an application-defined function to handle the extended notification procedure.

When you implement this function, the pointer you pass to the `DMRegisterExtendedNotifyProc` function should be a universal procedure pointer with the following type definition:

```
typedef (DMExtendedNotificationProcPtr)             DMExtendedNotificationUPP;
```

To create a universal procedure pointer for your application-defined function, you should use the `NewDMExtendedNotificationProc` macro as follows:

```
DMExtendedNotificationUPP  MyExtendedNotificationUPP;
MyExtendedNotificationUPP = NewDMExtendedNotificationProc
(MyExtendedNotificationCallback);
```

You can then pass `MyExtendedNotificationUPP` in the `notifyProc` parameter of the `DMRegisterExtendedNotifyProc` (page 1118) function. When you no longer need notifications, you should remove it using the `DMRemoveExtendedNotifyProc` (page 1120) function. You sould also dispose of the UPP using the `DisposeDMExtendedNotificationUPP` function:

```
DisposeDMExtendedNotificationUPP(MyExtendedNotificationUPP);
```

Using this call ensures that the call is made through a universal procedure pointer.

Special Considerations

Because this function may move or purge memory blocks or access handles, you cannot call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMNotificationProcPtr

```
typedef void (*DMNotificationProcPtr)
(
    AppleEvent * theEvent
);
```

If you name your function `MyDMNotificationProc`, you would declare it like this:

```
void MyDMNotificationProc (
    AppleEvent * theEvent
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMProfileListIteratorProcPtr

```
typedef void (*DMProfileListIteratorProcPtr)
(
    void * userData,
    DMListIndexType itemIndex,
    DMProfileListEntryPtr profileInfo
);
```

If you name your function `MyDMProfileListIteratorProc`, you would declare it like this:

```
void MyDMProfileListIteratorProc (
    void * userData,
    DMListIndexType itemIndex,
    DMProfileListEntryPtr profileInfo
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

Data Types

AVLocationRec

```
struct AVLocationRec {
    unsigned long locationConstant;
};
typedef struct AVLocationRec AVLocationRec;
typedef AVLocationRec * AVLocationPtr;
```

Fields

`locationConstant`

Reserved for future expansion. Set this field to zero.

Discussion

The function `DMGetGraphicInfoByAVID` (page 1106) uses the `AVLocationRec` structure to get information about graphic displays.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

AVPowerStatePtr

```
typedef VDPowerStateRec * AVPowerStatePtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

AVPowerStateRec

```
typedef VDPowerStateRec AVPowerStateRec;
```

Discussion

The functions `DMGetAVPowerState` (page 1100) and `DMSetAVPowerState` (page 1123) contain a parameter of type `AVPowerStatePtr`, which is a pointer to the `AVPowerStateRec` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DependentNotifyRec

```
struct DependentNotifyRec {
    ResType notifyType;
    ResType notifyClass;
    DisplayIDType notifyPortID;
    ComponentInstance notifyComponent;
    unsigned long notifyVersion;
    unsigned long notifyFlags;
    unsigned long notifyReserved;
    unsigned long notifyFuture;
};
typedef struct DependentNotifyRec DependentNotifyRec;
typedef DependentNotifyRec * DependentNotifyPtr;
```

Fields

`notifyType`
A value that specifies the type of engine, if any, that made the change. The Display Manager may set this field to zero.

`notifyClass`
A value specifying the class of change that occurred: for instance, color or screen size. This field uses a value supplied by the constant described under "Dependent Notification Constants" (page 1153) to specify the class of change that has occurred in a dependent display.

`notifyPortID`
Specifies which device was touched (`kInvalidDisplayID` specifies all or none).

`notifyComponent`
A value that indentifies the engine that made the change. The Display Manager may set this field to zero.

`notifyVersion`
Reserved for future expansion. The Display Manager sets this field to zero.

`notifyFlags`
Reserved for future expansion. The Display Manager sets this field to zero.

`notifyReserved`
Reserved for future expansion. The Display Manager sets this field to zero.

`notifyFuture`
Reserved for future expansion. The Display Manager sets this field to zero.

Discussion

The function `DMSendDependentNotification` (page 1122) uses the `notifyType` and `notifyClass` fields of the `DependentNotifyRec` structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DisplayListEntryRec

```
struct DisplayListEntryRec {
    GDHandle displayListEntryGDevice;
    DisplayIDType displayListEntryDisplayID;
    UInt32 displayListEntryIncludeFlags;
    UInt32 displayListEntryReserved1;
    UInt32 displayListEntryReserved2;
    UInt32 displayListEntryReserved3;
    UInt32 displayListEntryReserved4;
    UInt32 displayListEntryReserved5;
};
typedef struct DisplayListEntryRec DisplayListEntryRec;
typedef DisplayListEntryRec * DisplayListEntryPtr;
```

Fields

`displayListEntryGDevice`
A value of type GDHandle.

`displayListEntryDisplayID`
A value of type `DisplayIDType` that specifies the display ID.

`displayListEntryIncludeFlags`
A value of type `UInt32` that specifies the reason this entry was included.

`displayListEntryReserved1`
Reserved for future expansion. Set this field to zero.

`displayListEntryReserved2`
Reserved for future expansion. Set this field to zero.

`displayListEntryReserved3`
Reserved for future expansion. Set this field to zero.

`displayListEntryReserved4`
Reserved for future expansion. Set this field to zero.

`displayListEntryReserved5`
Reserved for future expansion. Set this field to zero.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMComponentListEntryRec

```
struct DMComponentListEntryRec {
    DisplayIDType itemID;
    Component itemComponent;
    ComponentDescription itemDescription;
    ResType itemClass;
    DMFidelityType itemFidelity;
    ResType itemSubClass;
    Point itemSort;
    unsigned long itemFlags;
    ResType itemReserved;
    unsigned long itemFuture1;
    unsigned long itemFuture2;
    unsigned long itemFuture3;
    unsigned long itemFuture4;
};
typedef struct DMComponentListEntryRec DMComponentListEntryRec;
typedef DMComponentListEntryRec * DMComponentListEntryPtr;
```

Fields

`itemID`

`itemComponent`

`itemDescription`

`itemClass`

`itemFidelity`

`itemSubClass`

`itemSort`

Reserved for future expansion. Set this field to zero.

`itemFlags`

Reserved for future expansion. Set this field to zero.

`itemReserved`

`itemFuture1`

Reserved for future expansion. Set this field to zero.

`itemFuture2`

Reserved for future expansion. Set this field to zero.

`itemFuture3`

Reserved for future expansion. Set this field to zero.

`itemFuture4`

Reserved for future expansion. Set this field to zero.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMComponentListIteratorUPP

```
typedef DMComponentListIteratorProcPtr DMComponentListIteratorUPP;
```

Discussion
For more information, see the description of the `DMComponentListIteratorProcPtr` (page 1132) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMDepthInfoBlockRec

```
struct DMDepthInfoBlockRec {
    unsigned long depthBlockCount;
    DMDepthInfoPtr depthVPBlock;
    unsigned long depthBlockFlags;
    unsigned long depthBlockReserved1;
    unsigned long depthBlockReserved2;
};
typedef struct DMDepthInfoBlockRec DMDepthInfoBlockRec;
typedef DMDepthInfoBlockRec * DMDepthInfoBlockPtr;
```

Fields

`depthBlockCount`
Specifies the number of mode depths available.

`depthVPBlock`
Array of `DMDepthInfoRec` (page 1142).

`depthBlockFlags`
Reserved for future expansion.

`depthBlockReserved1`
Reserved for future expansion.

`depthBlockReserved2`
Reserved for future expansion.

Discussion
When you call the function `DMGetIndexedDisplayModeFromList` (page 1107) , the Display Manager passes a pointer to a `DMDisplayModeListEntryRec` (page 1143) structure to your application. Its field `displayModeDepthBlockInfo` is a pointer to a `DMDepthInfoBlockRec` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMDepthInfoRec

```
struct DMDepthInfoRec {
    VDSwitchInfoPtr depthSwitchInfo;
    VPBlockPtr depthVPBlock;
    UInt32 depthFlags;
    UInt32 depthReserved1;
    UInt32 depthReserved2;
};
typedef struct DMDepthInfoRec DMDepthInfoRec;
typedef DMDepthInfoRec * DMDepthInfoPtr;
```

Fields

`depthSwitchInfo`
: A pointer to the structure `VDSwitchInfoRec`, which contains values that specify information on video switch modes and data.

`depthVPBlock`
: A pointer to the structure `VPBlock`, which supplies information about size, depth and format.

`depthFlags`
: Values from the video structure `VDVideoParametersInfoRec`, which specify color, size, and depth.

`depthReserved1`
: Reserved for future expansion.

`depthReserved2`
: Reserved for future expansion.

Discussion

This structure provides information that the structure `DMDepthInfoBlockRec` (page 1141) supplies to the function `DMGetIndexedDisplayModeFromList` (page 1107).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMDisplayListIteratorUPP

```
typedef DMDisplayListIteratorProcPtr DMDisplayListIteratorUPP;
```

Discussion

For more information, see the description of the `DMDisplayListIteratorProcPtr` (page 1133) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMDisplayModeListEntryRec

```
struct DMDisplayModeListEntryRec {
    UInt32 displayModeFlags;
    VDSwitchInfoPtr displayModeSwitchInfo;
    VDResolutionInfoPtr displayModeResolutionInfo;
    VDTimingInfoPtr displayModeTimingInfo;
    DMDepthInfoBlockPtr displayModeDepthBlockInfo;
    UInt32 displayModeVersion;
    StringPtr displayModeName;
    DMDisplayTimingInfoPtr displayModeDisplayInfo;
};
typedef struct DMDisplayModeListEntryRec DMDisplayModeListEntryRec;
typedef DMDisplayModeListEntryRec * DMDisplayModeListEntryPtr;
```

Fields

`displayModeFlags`
: A pointer to a video structure, `VDSwitchInfoRec`, which provides information you need to tell the driver how to switch into different configurations, bit depths, or resolutions. See the function `DMSetDisplayMode` (page 1125) for more information.

`displayModeSwitchInfo`
: A pointer to a `VDSwitchInfoRec` video structure, which provides information you need to tell the driver how to switch into different configurations, bit depths, or resolutions. See the function `DMSetDisplayMode` (page 1125) for more information.

`displayModeResolutionInfo`
: A pointer to a pointer to a `VDResolutionInfoRec` video structure, which provides information about horizontal pixels, maximum depth modes, and the vertical line of the specified display mode.

`displayModeTimingInfo`
: A pointer to a pointer to a `VDTimingInfoRec` video structure, which provides information about timing, format of the specified display mode.

`displayModeDepthBlockInfo`
: A pointer to a `DMDepthInfoBlockRec` (page 1141) structure, which provides information about available pixel formats and the `VPBlock`, including size and depth.

`displayModeVersion`
: The version of this structure. Currently it is version `kDisplayTimingInfoVersionOne`. See "Display Version Values" (page 1155) for more information.

`displayModeName`
: A string pointer giving the display mode name.

`displayModeDisplayInfo`
: A pointer to the `DMDisplayTimingInfoRec` (page 1144) data type. This data type supplies information about the quality and default values of the timing.

Discussion

The `DMDisplayModeListEntryRec` structure contains information about a display mode in a display mode list built by the function `DMNewDisplayModeList` (page 1117).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMDisplayModeListIteratorUPP

```
typedef DMDisplayModeListIteratorProcPtr DMDisplayModeListIteratorUPP;
```

Discussion
For more information, see the description of the `DMDisplayModeListIteratorProcPtr` (page 1133) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMDisplayTimingInfoRec

```
struct DMDisplayTimingInfoRec {
    UInt32 timingInfoVersion;
    UInt32 timingInfoAttributes;
    SInt32 timingInfoRelativeQuality;
    SInt32 timingInfoRelativeDefault;
    UInt32 timingInfoReserved[16];
};
typedef struct DMDisplayTimingInfoRec DMDisplayTimingInfoRec;
typedef DMDisplayTimingInfoRec * DMDisplayTimingInfoPtr;
```

Fields

`timingInfoVersion`
An unsigned 32 bit integer that shows the timing version. See "Display Version Values" (page 1155) for timing version values.

`timingInfoAttributes`
An unsigned 32 bit integer that the Display Manager sets to show timing attributes.

`timingInfoRelativeQuality`
A signed 32 bit integer whose flags the Display Manager sets to provide information on the quality of the timing.

`timingInfoRelativeDefault`
A signed 32 bit integer the Display Manager sets that specifies the relative default value of the timing.

`timingInfoReserved`
Reserved for future expansion.

Discussion
This structure supplies information about timing attributes, defaults and values to the structure `DMDisplayModeListEntryRec` (page 1143).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMExtendedNotificationUPP

```
typedef DMExtendedNotificationProcPtr DMExtendedNotificationUPP;
```

Discussion
For more information, see the description of the `DMExtendedNotificationProcPtr` (page 1134) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMFidelityType

```
typedef UInt32 DMFidelityType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMListIndexType

```
typedef unsigned long DMListIndexType;
```

Discussion
The function `DMGetIndexedDisplayModeFromList` (page 1107) uses this data type to supply a list of display modes from which you can obtain information about a specified display mode.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMListType

```
typedef void * DMListType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Displays.h`

DMMakeAndModelRec

```
struct DMMakeAndModelRec {
    ResType manufacturer;
    UInt32 model;
    UInt32 serialNumber;
    UInt32 manufactureDate;
    UInt32 makeReserved[4];
};
typedef struct DMMakeAndModelRec DMMakeAndModelRec;
typedef DMMakeAndModelRec * DMMakeAndModelPtr;
```

Fields

`manufacturer`
Represents the manufacturer of the specified display.

`model`
Represents the model name of the specified display.

`serialNumber`
Represents the serial number of the specified display.

`manufactureDate`
Represents the date of manufacture of the specified display.

`makeReserved`
Reserved for future expansion.

Discussion

This structure stores information about a specified monitor or display. If you need to keep track of configurations and user preferences, you can store that information in this structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMModalFilterUPP

```
typedef void * DMModalFilterUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Displays.h`

DMNotificationUPP

```
typedef DMNotificationProcPtr DMNotificationUPP;
```

Discussion

For more information, see the description of the `DMNotificationProcPtr` (page 1136) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In
Displays.h

DMProcessInfoPtr

```
typedef void * DMProcessInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
Displays.h

DMProfileListEntryRec

```
struct DMProfileListEntryRec {
    CMProfileRef profileRef;
    Ptr profileReserved1;
    Ptr profileReserved2;
    Ptr profileReserved3;
};
typedef struct DMProfileListEntryRec DMProfileListEntryRec;
typedef DMProfileListEntryRec * DMProfileListEntryPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
Displays.h

DMProfileListIteratorUPP

```
typedef DMProfileListIteratorProcPtr DMProfileListIteratorUPP;
```

Discussion
For more information, see the description of the DMProfileListIteratorProcPtr (page 1136) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Displays.h

Constants

Active Device Only Values

```
enum {
    dmOnlyActiveDisplays = true,
    dmAllDisplays = false
};
```

Constants

`dmOnlyActiveDisplays`

Returns a handle to the `GDevice` structure for an active device only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`dmAllDisplays`

Returns a handle to the `GDevice` structure for a device, active or not.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Discussion

The functions `DMGetFirstScreenDevice` (page 1104) and `DMGetNextScreenDevice` (page 1109) contain the parameter `activeOnly` which you can specify with an Active Device Constant.

Apple Event Notification Keywords

```
enum {
    kAESystemConfigNotice = 'cnfg',
    kAEDisplayNotice = 'dspl',
    kAEDisplaySummary = 'dsum',
    keyDMConfigVersion = 'dmcv',
    keyDMConfigFlags = 'dmcf',
    keyDMConfigReserved = 'dmcr',
    keyDisplayID = 'dmid',
    keyDisplayComponent = 'dmdc',
    keyDisplayDevice = 'dmdd',
    keyDisplayFlags = 'dmdf',
    keyDisplayMode = 'dmdm',
    keyDisplayModeReserved = 'dmmr',
    keyDisplayReserved = 'dmdr',
    keyDisplayMirroredId = 'dmmi',
    keyDeviceFlags = 'dddf',
    keyDeviceDepthMode = 'dddm',
    keyDeviceRect = 'dddr',
    keyPixMapRect = 'dpdr',
    keyPixMapHResolution = 'dphr',
    keyPixMapVResolution = 'dpvr',
    keyPixMapPixelType = 'dppt',
    keyPixMapPixelSize = 'dpps',
    keyPixMapCmpCount = 'dpcc',
    keyPixMapCmpSize = 'dpcs',
    keyPixMapAlignment = 'dppa',
    keyPixMapResReserved = 'dprr',
    keyPixMapReserved = 'dppr',
    keyPixMapColorTableSeed = 'dpct',
    keySummaryMenubar = 'dsmb',
    keySummaryChanges = 'dsch',
    keyDisplayOldConfig = 'dold',
    keyDisplayNewConfig = 'dnew'
};
```

Constants

`kAESystemConfigNotice`
Keyword for the Event ID for a Display Notice event.
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`kAEDisplayNotice`
Keyword for a required parameter to a Display Notice event.
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`kAEDisplaySummary`
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`keyDMConfigVersion`
Keyword for the descriptor structure describing the version number for this Display Notice event.
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`keyDMConfigFlags`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDMConfigReserved`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayID`
Keyword for the descriptor structure describing the display ID for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayComponent`
Unless you are disconnecting display components, this is for internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayDevice`
Keyword for the descriptor structure containing a handle to the `GDevice` structure for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayFlags`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayMode`
Keyword for the descriptor structure containing the `sResource` number from the video device for this display mode.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayModeReserved`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayReserved`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayMirroredId`
Keyword for the display this device is mirrored to.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDeviceFlags`
Keyword for the descriptor structure describing the attributes for the video device as maintained in the `gdFlags` field of the `GDevice` structure for the device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDeviceDepthMode`
Keyword for the descriptor structure describing the depth mode for the video device; that is, the value of the `gdMode` field in the `GDevice` structure for the device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDeviceRect`
Keyword for the descriptor structure describing the boundary rectangle of the video device; that is, the value of the `gdRect` field in the `GDevice` structure for the device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapRect`
Keyword for the descriptor structure describing the boundary rectangle into which QuickDraw can draw; that is, the bounds field in the `PixMap` structure for the `GDevice` structure for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapHResolution`
Keyword for the descriptor structure describing the horizontal resolution of the pixel image in the `PixMap` structure for the `GDevice` structure for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapVResolution`
Keyword for the descriptor structure describing the vertical resolution of the pixel image in the `PixMap` structure for the `GDevice` structure for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapPixelType`
Keyword for the descriptor structure describing the storage format for the pixel image on the device; that is, the value of the `pixelType` field in the `PixMap` structure for the `GDevice` structure for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapPixelSize`
Keyword for the descriptor structure describing the pixel depth for the device; that is, the value of the `pixelSize` field in the `PixMap` structure for the `GDevice` structure for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapCmpCount`
Keyword for the descriptor structure containing the number of components used to represent a color for a pixel; that is, the value of the `cmpCount` field in the `PixMap` structure for the `GDevice` structure for the device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapCmpSize`
Keyword for the descriptor structure describing the size in bits of each component for a pixel; that is, the value of the `cmpSize` field in the `PixMap` structure for the `GDevice` structure for the device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapAlignment`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapResReserved`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapReserved`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyPixMapColorTableSeed`
Keyword for the descriptor structure containing the value of the `ctSeed` field of the `ColorTable` structure for the `PixMap` structure for the `GDevice` structure for the video device.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keySummaryMenubar`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keySummaryChanges`
Reserved for future expansion. Internal use only.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayOldConfig`
Keyword for the descriptor structure describing the video device's previous state.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`keyDisplayNewConfig`
Keyword for the descriptor structure describing the video device's new state.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Discussion
The Display Manager sends an Apple event—the Display Notice event—to notify applications that it has changed the display environment. The keywords that specify the Display Notice event and its descriptor structures are described here.

Confirm Flags

```
enum {
    kForceConfirmBit = 0,
    kForceConfirmMask = (1 << kForceConfirmBit)
};
```

Constants
`kForceConfirmBit`
Indicates to force a confirm dialog.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kForceConfirmMask`
Use to set or test for a forced confirm dialog.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Dependent Notification Constants

```
enum {
    kDependentNotifyClassShowCursor = 'shcr',
    kDependentNotifyClassDriverOverride = 'ndrv',
    kDependentNotifyClassDisplayMgrOverride = 'dmgr',
    kDependentNotifyClassProfileChanged = 'prof'
};
```

Constants
`kDependentNotifyClassShowCursor`
The Display Manager sends an extended notification when a hidden cursor shows during a display unmirror.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDependentNotifyClassDriverOverride`
The Display Manager sends notification that a video driver has been overridden with a newer revision.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDependentNotifyClassDisplayMgrOverride`
The Display Manager sends notification that it has been upgraded with a newer revision.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDependentNotifyClassProfileChanged`
The Display Manager sends notification when the profile associated with a display changes.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Discussion
The function `DMSendDependentNotification` (page 1122) contains the parameter `notifyClass` which you can specify with a Dependent Notification Constant.

Display/Device ID Constants

The Display Manager uses these values to help with the configuration of the display.

```
enum {
    kDummyDeviceID = 0x00FF,
    kInvalidDisplayID = 0x0000,
    kFirstDisplayID = 0x0100
};
```

Constants
`kDummyDeviceID`
This is the ID of the dummy display, used when the last “real” display is removed.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kInvalidDisplayID`
This is the ID of the invalid display, which has been removed from the active display list.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kFirstDisplayID`
When your application sets this bit it asks the Display Manager to return the ID of the first display device on the active display list.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Display Gestalt Constants

```
enum {
    kDisplayGestaltDisplayCommunicationAttr = 'comm',
    kDisplayGestaltForbidI2CMask = (1 << 0),
    kDisplayGestaltUseI2CPowerMask = (1 << 1),
    kDisplayGestaltCalibratorAttr = 'cali',
    kDisplayGestaltBrightnessAffectsGammaMask = (1 << 0),
    kDisplayGestaltViewAngleAffectsGammaMask = (1 << 1)
};
```

Display Mode Flags

The structure `DMDisplayModeListEntryRec` uses these values for its `displayModeFlags` field.

```
enum {
    kDisplayModeListNotPreferredBit = 0,
    kDisplayModeListNotPreferredMask = (1 << kDisplayModeListNotPreferredBit)
};
```

Constants

`kDisplayModeListNotPreferredBit`
Indicates there is a better timing available and that this timing should be shown only if the user wants to see all options.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDisplayModeListNotPreferredMask`
(1 `kDisplayModeListNotPreferredBit`)

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Display Version Values

```
enum {
    kDisplayTimingInfoVersionZero = 1,
    kDisplayTimingInfoReservedCountVersionZero = 16,
    kDisplayModeEntryVersionZero = 0,
    kDisplayModeEntryVersionOne = 1
};
```

Constants

`kDisplayTimingInfoVersionZero`
This relative information is always `NULL` in this version.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDisplayTimingInfoReservedCountVersionZero`
This relative information is always `NULL` in this version.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDisplayModeEntryVersionZero`
This relative information is always `NULL` in this version.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDisplayModeEntryVersionOne`
This relative information is always `NULL` in this version.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Discussion
These values supply information to the structure `DMDisplayModeListEntryRec` (page 1143).

Fidelity Check Constants

```
enum {
    kNoFidelity = 0,
    kMinimumFidelity = 1,
    kDefaultFidelity = 500,
    kDefaultManufacturerFidelity = 1000
};
```

Get Name By AVID Mask

```
enum {
    kDMSupressNumbersMask = (1 << 0),
    kDMForceNumbersMask = (1 << 1),
    kDMSupressNameMask = (1 << 2)
};
```

Constants
`kDMSupressNumbersMask`
If the bit specified by this mask is set, the numbers are suppressed and only names are returned.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMForceNumbersMask`
If the bit specified by this mask is set, the numbers are forced to always be shown–even on single display configs.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMSupressNameMask`
If the bit specified by this mask is set, the names are suppressed and only numbers are returned.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Include Masks

```
enum {
    kIncludeOnlineActiveDisplaysMask = (1 << 0),
    kIncludeOnlineDisabledDisplaysMask = (1 << 1),
    kIncludeOfflineDisplaysMask = (1 << 2),
    kIncludeOfflineDummyDisplaysMask = (1 << 3),
    kIncludeHardwareMirroredDisplaysMask = (1 << 4)
};
```

Item Flags

```
enum {
    kComponentListNotPreferredBit = 0,
    kComponentListNotPreferredMask = (1 << kComponentListNotPreferredBit)
};
```

Mode List Masks

```
enum {
    kDMModeListIncludeAllModesMask = (1 << 0),
    kDMModeListIncludeOfflineModesMask = (1 << 1),
    kDMModeListExcludeDriverModesMask = (1 << 2),
    kDMModeListExcludeDisplayModesMask = (1 << 3),
    kDMModeListExcludeCustomModesMask = (1 << 4),
    kDMModeListPreferStretchedModesMask = (1 << 5),
    kDMModeListPreferSafeModesMask = (1 << 6)
};
```

Constants

`kDMModeListIncludeAllModesMask`
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`kDMModeListIncludeOfflineModesMask`
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`kDMModeListExcludeDriverModesMask`
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`kDMModeListExcludeDisplayModesMask`
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`kDMModeListExcludeCustomModesMask`
Available in Mac OS X v10.0 and later.
Declared in `Displays.h`.

`kDMModeListPreferStretchedModesMask`

Prefer modes that are stretched over modes that are letterboxed when setting kDisplayModeListNotPreferredBit

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMModeListPreferSafeModesMask`

Prefer modes that are safe over modes that are not when setting kDisplayModeListNotPreferredBit

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Name Flags

```
enum {
    kSuppressNumberBit = 0,
    kSuppressNumberMask = 1,
    kForceNumberBit = 1,
    kForceNumberMask = 2,
    kSuppressNameBit = 2,
    kSuppressNameMask = 4
};
```

New Engine List Constants

```
enum {
    kAnyPanelType = 0,
    kAnyEngineType = 0,
    kAnyDeviceType = 0,
    kAnyPortType = 0
};
```

Notification Messages

```
enum {
    kDMNotifyRequestConnectionProbe = 0,
    kDMNotifyInstalled = 1,
    kDMNotifyEvent = 2,
    kDMNotifyRemoved = 3,
    kDMNotifyPrep = 4,
    kDMNotifyExtendEvent = 5,
    kDMNotifyDependents = 6,
    kDMNotifySuspendConfigure = 7,
    kDMNotifyResumeConfigure = 8,
    kDMNotifyRequestDisplayProbe = 9,
    kDMNotifyDisplayWillSleep = 10,
    kDMNotifyDisplayDidWake = 11,
    kExtendedNotificationProc = (1L << 16)
};
```

Constants

`kDMNotifyRequestConnectionProbe`

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyInstalled`

The Display Manager provides this message during a callback function to if your application has installed an extended notification procedure pointer for the first time. The Display Manager provides this message in the `notifyData` parameter of `DMSendDependentNotification` (page 1122).

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyEvent`
The Display Manager provides this message when an Apple event update occurs, after a display configuration change is made. This is the only time non-extended notifications are called.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyRemoved`
The Display Manager provides this message when the function `DMSendDependentNotification` (page 1122) is called on your function.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyPrep`
Before passing `kDMSNotifyRemoved`, the Display Manager provides this message to indicate that it is about to begin to configure. Calling `DMSendDependentNotification` (page 1122) tells the Display Manager to send this message.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyExtendEvent`
Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyDependents`
The Display Manager provides this message to `DMSendDependentNotification` (page 1122).

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifySuspendConfigure`
The Display Manager passes this selector to notify your UPP that configuration is temporarily suspended. For instance, if a video game makes a temporary change to the display configuration, the game is expected to resume configuration and restore video before allowing other applications to access the screen.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyResumeConfigure`
The Display Manager passes this selector to notify your application when previously suspended configuration is resumed. Your application can then replace windows and icons, and change depth mode if necessary.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyRequestDisplayProbe`
Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDMNotifyDisplayWillSleep`
This selector is only available in Mac OS X.

Available in Mac OS X v10.2 and later.

Declared in `Displays.h`.

`kDMNotifyDisplayDidWake`
This selector is only available in Mac OS X.

Available in Mac OS X v10.2 and later.

Declared in `Displays.h`.

`kExtendedNotificationProc`
Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Discussion
Display Manager functions needed for dependency notification and event processing use the notification message selectors in extended application-defined functions. `DMRegisterExtendedNotifyProc` (page 1118) gets all these messages. Applications should update all information about the display configurations at this point.

Notification Types

```
enum {
    kFullNotify = 0,
    kFullDependencyNotify = 1
};
```

Constants
`kFullNotify`
The Display Manager sets this bit to provide the major Apple notification event.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kFullDependencyNotify`
The Display Manager sets this bit to provide notification only to those applications that need to know about interrelated functionality. It is used for updating the user interface.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Discussion
The function `DMSendDependentNotification` (page 1122) uses these values in the `notifyType` parameter.

Panel List Flags

```
enum {
    kAllowDuplicatesBit = 0
};
```

Port List Flags

```
enum {
    kPLIncludeOfflineDevicesBit = 0
};
```

Constants

`kPLIncludeOfflineDevicesBit`

Should offline devices be put into the port list (such as dummy display)

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Reserved Count Constants

```
enum {
    kMakeAndModelReservedCount = 4
};
```

Constants

`kMakeAndModelReservedCount`

Indicates the number of reserved fields.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Summary Change Flags

```
enum {
    kBeginEndConfigureBit = 0,
    kMovedDisplayBit = 1,
    kSetMainDisplayBit = 2,
    kSetDisplayModeBit = 3,
    kAddDisplayBit = 4,
    kRemoveDisplayBit = 5,
    kNewDisplayBit = 6,
    kDisposeDisplayBit = 7,
    kEnabledDisplayBit = 8,
    kDisabledDisplayBit = 9,
    kMirrorDisplayBit = 10,
    kUnMirrorDisplayBit = 11
};
```

Switch Flags

```
enum {
    kNoSwitchConfirmBit = 0,
    kDepthNotAvailableBit = 1,
    kShowModeBit = 3,
    kModeNotResizeBit = 4,
    kNeverShowModeBit = 5
};
```

Constants

`kNoSwitchConfirmBit`

If the Display Manager sets this bit the display mode is required to function correctly. Your application does not need to provide confirmation if the user switches to this mode.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kDepthNotAvailableBit`

If the Display Manager sets this bit the pixel depth of the specified device is not available for the specified display mode.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kShowModeBit`

If the Display Manager sets this bit your application should display this mode to the user, even though it may require confirmation.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kModeNotResizeBit`

If the Display Manager sets this bit you should not use this mode to resize a display; this mode drives a different connector in cards than in a built-in display.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

`kNeverShowModeBit`

If the Display Manager sets this bit you should not show the mode in the user interface.

Available in Mac OS X v10.0 and later.

Declared in `Displays.h`.

Discussion

In its `switchFlags` parameter, the function `DMCheckDisplayMode` (page 1094) returns a pointer to a long integer that specifies flags in two of its bits. The constants represent bits that are set to 1. These bits are set by the Display Manager, not your application

Result Codes

The table below lists the result codes that are specific to the Display Manager.

Result Code	Value	Description
`kDMGenErr`	-6220	An indeterminate error occurred. Available in Mac OS X v10.0 and later.
`kDMMirroringOnAlready`	-6221	Video mirroring is already enabled. Available in Mac OS X v10.0 and later.
`kDMWrongNumberOfDisplays`	-6222	Wrong number of displays. Available in Mac OS X v10.0 and later.
`kDMMirroringBlocked`	-6223	Video is blocked. Available in Mac OS X v10.0 and later.
`kDMCantBlock`	-6224	Video mirroring is already enabled and can't be blocked; use `DMUnMirrorDevice`, then call `DMBlockMirroring` again. Available in Mac OS X v10.0 and later.
`kDMMirroringNotOn`	-6225	Video mirroring is not currently enabled. Available in Mac OS X v10.0 and later.
`kSysSWTooOld`	-6226	Some piece of system software is too old for the Display Manager to operate. Available in Mac OS X v10.0 and later.
`kDMSWNotInitializedErr`	-6227	The required pieces of system software are not initialized. Available in Mac OS X v10.0 and later.
`kDMDriverNotDisplayMgrAwareErr`	-6228	The video driver for the display does not support the Display Manager. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kDMNotFoundErr	-6229	Available in Mac OS X v10.0 and later.
kDMDisplayNotFoundErr	-6229	There are no GDevice structures for displays in the device list. Available in Mac OS X v10.0 and later.
kDMDisplayAlreadyInstalledErr	-6230	The display is already in the device list and can't be added. Available in Mac OS X v10.0 and later.
kDMNoDeviceTableclothErr	-6231	Available in Mac OS X v10.0 and later.
kDMMainDisplayCannotMoveErr	-6231	Available in Mac OS X v10.0 and later.
kDMFoundErr	-6232	Item found Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Display Manager Version selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Font Manager Reference

Framework: ApplicationServices/ApplicationServices.h

Declared in Fonts.h

Overview

As of Mac OS X version 10.5, all Font Manager functions but three (`FMFontGetCGFontRefFromFontFamilyInstance` (page 1178), `FMGetATSFontRefFromFont` (page 1179), and `FMGetFontFromATSFontRef` (page 1186)) are deprecated. Most were deprecated in Mac OS X v10.4. The Font Manager was the font management API for the QuickDraw framework, which is now deprecated.

There are several alternatives that provide better compatibility with the rest of Mac OS X than using QuickDraw font functions. You should consider the following:

- For drawing and measuring text, use Core Text on Mac OS X v10.5 and later to render text directly through a Quartz (Core Image) graphics context. See *Core Text Programming Guide* and *Core Text Reference Collection*. On Mac OS X v10.4 and earlier, you can use the Appearance Manager API or the ATSUI API. See *Appearance Manager Reference*, *ATSUI Programming Guide*, and *ATSUI Reference*.
- For accessing information on fonts tracked by the operating system, use Core Text on Mac OS X v10.5 and later. See *Core Text Programming Guide* and *Core Text Reference Collection*. On Mac OS X v10.4 and earlier, use the ATS for Fonts API. See *Apple Type Services for Fonts Programming Guide* and *Apple Type Services for Fonts Reference*.
- For accessing and modifying information on fonts in a Quartz graphics context, use the Quartz API. See *Quartz 2D Programming Guide* and *Quartz 2D Reference Collection*.

The Font Manager API was used to manage the fonts your application uses to display and print text. The Font Manager was used to determine the characteristics of a font, change certain font settings, favor outline fonts over bitmapped fonts, and manipulate fonts in memory.

Functions by Task

Activating and Deactivating Fonts

`FMGetGeneration` (page 1189) Deprecated in Mac OS X v10.5
Retrieves the value of the generation count. (Deprecated. Use `ATSGetGeneration` (page 678) instead.)

`FMActivateFonts` (page 1172) Deprecated in Mac OS X v10.4
Activates one or more fonts. (Deprecated. Use `ATSFontActivateFromFileReference` (page 647) instead.)

FMDeactivateFonts (page 1176) Deprecated in Mac OS X v10.4
Deactivates one or more fonts. (Deprecated. Use ATSFontDeactivate (page 651) instead.)

Accessing Font Objects

FMGetFontContainer (page 1179) Deprecated in Mac OS X v10.4
Obtains the file that contains data for a font. (Deprecated. Use ATSFontGetContainer (page 662) instead.)

FMGetFontFormat (page 1185) Deprecated in Mac OS X v10.4
Obtains the format identifier of a font.

FMGetFontGeneration (page 1187) Deprecated in Mac OS X v10.4
Obtains the generation count of a font. (Deprecated. Use ATSFontGetGeneration (page 665) instead.)

FMGetFontTable (page 1188) Deprecated in Mac OS X v10.4
Retrieves all or part of a data table for a font. (Deprecated. Use ATSFontGetTable (page 668) or CTFontCopyTable instead.)

FMGetFontTableDirectory (page 1189) Deprecated in Mac OS X v10.4
Obtains the table directory for a font.

Accessing Font Containers

FMGetFontContainerFromFontFamilyInstance (page 1180) Deprecated in Mac OS X v10.4
Obtains the font container associated with a font family instance. (Deprecated. Use ATSFontGetContainer (page 662) instead.)

FMGetFontFamilyResource (page 1184) Deprecated in Mac OS X v10.4
Obtains the font family resource for a font family. (Deprecated. Use ATSFontGetFontFamilyResource (page 664) instead.)

Accessing Font Family Objects

FMGetFontFamilyFromName (page 1181) Deprecated in Mac OS X v10.4
Returns the font family reference associated with a standard QuickDraw name. (Deprecated. Use ATSFontFamilyFindFromName (page 652) instead.)

FMGetFontFamilyGeneration (page 1182) Deprecated in Mac OS X v10.4
Obtains the generation count of a font family. (Deprecated. Use ATSFontFamilyGetGeneration (page 654) instead.)

FMGetFontFamilyName (page 1183) Deprecated in Mac OS X v10.4
Obtains the font family name associated with a font family reference. (Deprecated. Use ATSFontFamilyGetName (page 654) instead.)

FMGetFontFamilyTextEncoding (page 1185) Deprecated in Mac OS X v10.4
Obtains the text encoding used by a font family. (Deprecated. Use ATSFontFamilyGetEncoding (page 653) instead.)

Enumerating Font Data

`FMCreateFontFamilyInstanceIterator` (page 1173) Deprecated in Mac OS X v10.4
Creates a font family instance iterator that your application can use to access the member fonts associated with a font family.

`FMCreateFontFamilyIterator` (page 1174) Deprecated in Mac OS X v10.4
Creates a font family iterator that your application can use to access font family objects. (Deprecated. Use `ATSFontFamilyIteratorCreate` (page 655) instead.)

`FMCreateFontIterator` (page 1175) Deprecated in Mac OS X v10.4
Creates an iterator that your application can use to access fonts. (Deprecated. Use `ATSFontIteratorCreate` (page 671) instead.)

`FMDisposeFontFamilyInstanceIterator` (page 1176) Deprecated in Mac OS X v10.4
Disposes of a font family instance iterator.

`FMDisposeFontFamilyIterator` (page 1177) Deprecated in Mac OS X v10.4
Disposes of the contents of a font family iterator. (Deprecated. Use `ATSFontFamilyIteratorRelease` (page 658) instead.)

`FMDisposeFontIterator` (page 1177) Deprecated in Mac OS X v10.4
Disposes of a font iterator. (Deprecated. Use `ATSFontIteratorRelease` (page 673) instead.)

`FMGetNextFont` (page 1190) Deprecated in Mac OS X v10.4
Obtains the next font reference. (Deprecated. Use `ATSFontIteratorNext` (page 672) instead.)

`FMGetNextFontFamily` (page 1190) Deprecated in Mac OS X v10.4
Obtains the next font family reference. (Deprecated. Use `ATSFontFamilyIteratorNext` (page 657) instead.)

`FMGetNextFontFamilyInstance` (page 1191) Deprecated in Mac OS X v10.4
Obtains the next instance associated with a font family reference.

`FMResetFontFamilyInstanceIterator` (page 1192) Deprecated in Mac OS X v10.4
Resets the a font family instance iterator to the beginning of the iteration for the specified font family.

`FMResetFontFamilyIterator` (page 1193) Deprecated in Mac OS X v10.4
Resets a font family iterator to the beginning of the iteration. (Deprecated. Use `ATSFontFamilyIteratorReset` (page 658) instead.)

`FMResetFontIterator` (page 1194) Deprecated in Mac OS X v10.4
Resets a font iterator to the beginning of the iteration. (Deprecated. Use `ATSFontIteratorReset` (page 674) instead.)

Converting Font Data

`FMFontGetCGFontRefFromFontFamilyInstance` (page 1178)
Obtains the Quartz font associated with a typeface from a QuickDraw font family.

`FMGetATSFontRefFromFont` (page 1179)
Obtains the ATS font reference associated with a font object.

`FMGetFontFromATSFontRef` (page 1186)
Obtains the font object associated with an ATS font reference.

`FMGetATSFontFamilyRefFromFontFamily` (page 1179) Deprecated in Mac OS X v10.4
Obtains the ATS font family reference associated with a font family object.

`FMGetFontFamilyFromATSFontFamilyRef` (page 1181) Deprecated in Mac OS X v10.4
Obtains the font family associated with an ATS font family reference.

`FMGetFontFamilyInstanceFromFont` (page 1182) Deprecated in Mac OS X v10.4
Finds the font family reference and standard QuickDraw style associated with a font.

`FMGetFontFromFontFamilyInstance` (page 1187) Deprecated in Mac OS X v10.4
Obtains the font reference associated with a standard QuickDraw style and font family. (Deprecated. Use `CTFontCreateWithQuickdrawInstance` instead.)

Getting Font Information

`FetchFontInfo` (page 1171) Deprecated in Mac OS X v10.4
Obtains the information for a specific font. (Deprecated. There is no replacement function.)

`FMSwapFont` (page 1195) Deprecated in Mac OS X v10.4
Returns a pointer to the font output structure for a specified font. (Deprecated. There is no replacement function.)

`FontMetrics` (page 1195) Deprecated in Mac OS X v10.4
Obtains fractional measurements for the font, size, and style specified in the current graphics port. (Deprecated. There is no replacement function.)

`GetFNum` (page 1197) Deprecated in Mac OS X v10.4
Obtains the font family ID for a specified font family name. (Deprecated. Use `ATSFontFamilyFindFromName` (page 652) instead.)

`GetFontName` (page 1198) Deprecated in Mac OS X v10.4
Obtains the name of a font family that has a specified family ID number. (Deprecated. Use `ATSFontFamilyGetName` (page 654) instead.)

`OutlineMetrics` (page 1201) Deprecated in Mac OS X v10.4
Obtains font measurements for a block of text to be drawn in a specified outline font. (Deprecated. There is no replacement function.)

`RealFont` (page 1203) Deprecated in Mac OS X v10.4
Determines whether a font is available or is intended for use in a specified size. (Deprecated. There is no replacement function.)

Working With Outline Fonts

`GetOutlinePreferred` (page 1198) Deprecated in Mac OS X v10.4
Obtains the current preference for whether outline or bitmapped fonts are returned when the Font Manager receives a font request. (Deprecated. There is no replacement function.)

`GetPreserveGlyph` (page 1199) Deprecated in Mac OS X v10.4
Determines whether the Font Manager preserves the shapes of glyphs from outline fonts. (Deprecated. There is no replacement function.)

`IsOutline` (page 1200) Deprecated in Mac OS X v10.4
Determines whether the specified scaling factors will cause the Font Manager to choose an outline font for the current graphics port. (Deprecated. There is no replacement function.)

`SetOutlinePreferred` (page 1206) Deprecated in Mac OS X v10.4
Sets the preference for whether to use bitmapped or outline fonts when both kinds of fonts are available. (Deprecated. There is no replacement function.)

SetPreserveGlyph (page 1206) Deprecated in Mac OS X v10.4
Temporarily changes the default behavior of the Font Manager, so that it does not scale oversized glyphs. (Deprecated. There is no replacement function.)

Working with Antialiased Text

IsAntiAliasedTextEnabled (page 1199) Deprecated in Mac OS X v10.4
Checks whether antialiased text is enabled. (Deprecated. There is no replacement function.)

SetAntiAliasedTextEnabled (page 1204) Deprecated in Mac OS X v10.4
Enables or disables antialiased text for an application. (Deprecated. There is no replacement function.)

Working With Font Measurements and Scaling

QDTextBounds (page 1203) Deprecated in Mac OS X v10.4
Obtains a rectangle that specifies the bounds of QuickDraw text. (Deprecated. There is no replacement function.)

SetFractEnable (page 1204) Deprecated in Mac OS X v10.4
Enables or disables fractional glyph widths. (Deprecated. There is no replacement function.)

SetFScaleDisable (page 1205) Deprecated in Mac OS X v10.4
Enables or disables the computation of font scaling factors by the Font Manager for bitmapped glyphs. (Deprecated. There is no replacement function.)

Using the Current, System, and Application Fonts

GetAppFont (page 1196) Deprecated in Mac OS X v10.4
Returns the font family ID of the current application font. (Deprecated. There is no replacement function.)

GetDefFontSize (page 1197) Deprecated in Mac OS X v10.4
Determines the default size of the system font. (Deprecated. There is no replacement function.)

GetSysFont (page 1199) Deprecated in Mac OS X v10.4
Obtains the font family ID of the current system font. (Deprecated. There is no replacement function.)

Functions

FetchFontInfo

Obtains the information for a specific font. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr FetchFontInfo (
   SInt16 fontID,
   SInt16 fontSize,
   SInt16 fontStyle,
   FontInfo *info
);
```

Parameters

fontID

A signed, 16-bit integer that specifies the font ID of the font whose information you want to obtain.

fontSize

A signed, 16-bit integer that specifies the font size of the font whose information you want to obtain.

fontStyle

A signed, 16-bit integer that specifies the font style of the font whose information you want to obtain.

info

On output, points to a font information structure that contains measurement information (ascent, descent, width, and leading) for the specified font.

Return Value

A result code. See "Font Manager Result Codes" (page 1230).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMActivateFonts

Activates one or more fonts. (Deprecated in Mac OS X v10.4. Use `ATSFontActivateFromFileReference` (page 647) instead.)

```
OSStatus FMActivateFonts (
   const FSSpec *iFontContainer,
   const FMFilter *iFilter,
   void *iRefCon,
   OptionBits iOptions
);
```

Parameters

iFontContainer

A pointer to the file specification of the file that contains the font data you want to activate. You can specify a directory or an individual font file.

iFilter

A pointer to a filter specification. This parameter is currently reserved for future use, so you should pass `NULL`.

iRefCon

An arbitrary 32-bit value specified by your application. This parameter is currently reserved for future use, so you should pass `NULL`.

iOptions

A value that specifies the scope to which the function applies. If you want the Font Manager to make the fonts visible only to your application, use the constant `kFMLocalActivationContext`. If you want the Font Manager to make fonts visible to all applications installed on the system, use the constant `kFMGlobalActivationContext`. See Activation Contexts (page 1224) for more information on these constants.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMCreateFontFamilyInstanceIterator

Creates a font family instance iterator that your application can use to access the member fonts associated with a font family. (Deprecated in Mac OS X v10.4.)

```
OSStatus FMCreateFontFamilyInstanceIterator (
   FMFontFamily iFontFamily,
   FMFontFamilyInstanceIterator *ioIterator
);
```

Parameters

iFontFamily

A reference to the font family you want to access.

ioIterator

A pointer to a structure of type `FMFontFamilyInstanceIterator`. On input, pass a pointer to an uninitialized structure. On output, its contents may have been changed and may include references to other data structures allocated by the system to maintain the structure's state. The iterator is positioned before the first member font of the font family. When you no longer need the font family instance iterator, you should call the function `FMDisposeFontFamilyInstanceIterator` to release the auxiliary data and memory allocated by the system.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
A font family instance iterator is an opaque data structure used by the Font Manager to keep track of an iteration over currently active font family instances. A font family instance is a typeface and a size—an entry from the font association table.

When the font family iterator is initialized, it does not yet reference a font family instance. Do not attempt to modify the contents of a font family instance iterator.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMCreateFontFamilyIterator

Creates a font family iterator that your application can use to access font family objects. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyIteratorCreate` (page 655) instead.)

```
OSStatus FMCreateFontFamilyIterator (
   const FMFilter *iFilter,
   void *iRefCon,
   OptionBits iOptions,
   FMFontFamilyIterator *ioIterator
);
```

Parameters

iFilter

A pointer to a filter specification. Pass `NULL` if you want to access all font family objects within the scope of your iteration. Otherwise, you can use this parameter to restrict the scope of the iteration to the font families that match a generation count or criteria you specify in a custom filter function. Pass the filter selector constant `kFMGenerationFilterSelector` to select a generation filter or the constant `kFMFontFamilyCallbackFilterSelector` to select a custom filter. See `FMFilterSelector` in the ATS Types Reference for more information on these constants.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. If you are not using a custom filter function, pass `NULL`.

iOptions

A value that specifies the scope to which the font family iterator applies. If you want the Font Manager to apply the font family iterator only to the fonts accessible from your application use the `kFMLocalIterationScope` constant. If you want the Font Manager to apply the font family iterator to all fonts registered with the system use the constant `kFMGlobalIterationScope`. See Activation Contexts (page 1224) for more information on these constants.

ioIterator

A pointer to a structure of type `FMFontFamilyIterator`. On input, pass a pointer to an uninitialized structure. On output, the structure's contents may have been changed and may include references to other data structures allocated by the system to maintain the structure's state. When you no longer need the font family iterator, you should call the function `FMDisposeFontFamilyIterator` to release the auxiliary data and memory allocated by the system.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
A font family iterator is an opaque data structure used by the Font Manager to keep track of an iteration over currently active font families. When the font family iterator is initialized, it does not yet reference a font family. Do not attempt to modify the contents of a font family iterator.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMCreateFontIterator

Creates an iterator that your application can use to access fonts. (Deprecated in Mac OS X v10.4. Use `ATSFontIteratorCreate` (page 671) instead.)

```
OSStatus FMCreateFontIterator (
   const FMFilter *iFilter,
   void *iRefCon,
   OptionBits iOptions,
   FMFontIterator *ioIterator
);
```

Parameters

iFilter

A pointer to font filter specification. Pass `NULL` if you want to access all font objects within the scope of your iteration. Otherwise, you can use this parameter to restrict the scope of the iteration to font information that matches a technology, font container, or criteria you specify in a custom filter function. Pass the filter selector constant `kFMFontTechnologyFilterSelector` to select a font technology filter, the constant `kFMFontContainerFilterSelector` to select a font container filter, or the constant `kFMFontCallbackFilterSelector` to select a custom filter. See `FMFilterSelector` in the ATS Types Reference for more information on these constants.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. If are not using a custom filter function, pass `NULL`.

iOptions

A value that specifies the scope to which the font iterator applies. If you want the Font Manager to apply the font iterator only to the fonts accessible from your application use the `kFMLocalIterationScope` constant. If you want the Font Manager to apply the font iterator to all fonts registered with the system use the constants `kFMGlobalIterationScope`. See Activation Contexts (page 1224) for more information on these constants.

ioIterator

A pointer to a structure of type `FMFontIterator`. On input, pass a pointer to an uninitialized structure. On output, the structure's contents may have been changed and may include references to other data structures allocated by the system to maintain the structure's state. When you no longer need the font iterator, you should call the function `FMDisposeFontIterator` to release the auxiliary data and memory allocated by the system.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
A font iterator is an opaque structure used by the Font Manager to maintain font information in the context of the current application process. When the font iterator is initialized, it is not yet positioned on a font object. You should not attempt to modify the contents of a font iterator.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Fonts.h`

FMDeactivateFonts

Deactivates one or more fonts. (Deprecated in Mac OS X v10.4. Use `ATSFontDeactivate` (page 651) instead.)

```
OSStatus FMDeactivateFonts (
   const FSSpec *iFontContainer,
   const FMFilter *iFilter,
   void *iRefCon,
   OptionBits iOptions
);
```

Parameters

iFontContainer

A pointer to the file specification of the file that contains the font data you want to deactivate. You can specify a directory or an individual font file.

iFilter

A pointer to a filter specification. This parameter is currently reserved for future use, so you should pass `NULL`.

iRefCon

An arbitrary 32-bit value specified by your application. This parameter is currently reserved for future use, so you should pass `NULL`.

iOptions

A value that specifies the scope to which the function applies. This parameter is currently reserved for future use, so you should pass `NULL`.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMDisposeFontFamilyInstanceIterator

Disposes of a font family instance iterator. (Deprecated in Mac OS X v10.4.)

```
OSStatus FMDisposeFontFamilyInstanceIterator (
   FMFontFamilyInstanceIterator *ioIterator
);
```

Parameters

ioIterator

A pointer to a font family instance iterator you created with the function `FMCreateFontFamilyInstanceIterator` (page 1173). If you try to use the font family instance iterator after disposing of its contents through this function, the Font Manager returns an error code to your application.

Return Value

A result code. See "Font Manager Result Codes" (page 1230).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMDisposeFontFamilyIterator

Disposes of the contents of a font family iterator. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyIteratorRelease` (page 658) instead.)

```
OSStatus FMDisposeFontFamilyIterator (
   FMFontFamilyIterator *ioIterator
);
```

Parameters

ioIterator

A pointer to a font family iterator you created with the function `FMCreateFontFamilyIterator` (page 1174). If you try to use the font family iterator after disposing of its contents through this function, the Font Manager returns an error code to your application.

Return Value

A result code. See "Font Manager Result Codes" (page 1230).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMDisposeFontIterator

Disposes of a font iterator. (Deprecated in Mac OS X v10.4. Use `ATSFontIteratorRelease` (page 673) instead.)

```
OSStatus FMDisposeFontIterator (
   FMFontIterator *ioIterator
);
```

Parameters

ioIterator

A pointer to a font iterator you created with the function `FMCreateFontIterator` (page 1175). If you try to use the font iterator after disposing of its contents through this function, the Font Manager returns an error code to your application.

Return Value

A result code. See "Font Manager Result Codes" (page 1230).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMFontGetCGFontRefFromFontFamilyInstance

Obtains the Quartz font associated with a typeface from a QuickDraw font family.

```
OSStatus FMFontGetCGFontRefFromFontFamilyInstance (
   FMFontFamily iFontFamily,
   FMFontStyle iStyle,
   CGFontRef *oFont,
   FMFontStyle *oStyle
);
```

Parameters

iFontFamily

A QuickDraw font family.

iStyle

A QuickDraw font style.

oFont

A pointer to a Quartz font reference. On output, points to the Quartz font reference for the specified font family and style. You are responsible for allocating the memory for the Quartz font reference.

oStyle

On output, a pointer to an intrinsic font style. If a font object isn't found that matches the font family reference and font style you specify, the function returns the QuickDraw style that matches most closely.

Return Value

A result code. See "Font Manager Result Codes" (page 1230). If a font reference and intrinsic style are not found, the function returns a value of `kFMInvalidFontErr`.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetATSFontFamilyRefFromFontFamily

Obtains the ATS font family reference associated with a font family object. (Deprecated in Mac OS X v10.4.)

```
ATSFontFamilyRef FMGetATSFontFamilyRefFromFontFamily (
   FMFontFamily iFamily
);
```

Parameters

iFamily
A font family reference.

Return Value
The `ATSFontFamilyRef` associated with the font family object.

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetATSFontRefFromFont

Obtains the ATS font reference associated with a font object.

```
ATSFontRef FMGetATSFontRefFromFont (
   FMFont iFont
);
```

Parameters

iFont
A font reference.

Return Value
The `ATSFontRef` associated with the font object.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontContainer

Obtains the file that contains data for a font. (Deprecated in Mac OS X v10.4. Use `ATSFontGetContainer` (page 662) instead.)

```
OSStatus FMGetFontContainer (
   FMFont iFont,
   FSSpec *oFontContainer
);
```

Parameters

iFont

A font reference.

oFontContainer

On output, a pointer to the file specification of the file that contains the font data.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
You can pass the file specification returned by this function to the Resource Manager or File Manager to obtain the actual font data. However, if the font is an LWFN-class font, the outline data is located in a separate file from the font suitcase. The function `FMGetFontContainer` obtains the font suitcase. Your application is responsible for finding the individual outline files.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontContainerFromFontFamilyInstance

Obtains the font container associated with a font family instance. (Deprecated in Mac OS X v10.4. Use `ATSFontGetContainer` (page 662) instead.)

```
OSStatus FMGetFontContainerFromFontFamilyInstance (
   FMFontFamily iFontFamily,
   FMFontStyle iStyle,
   FMFontSize iFontSize,
   FSSpec *oFontContainer
);
```

Parameters

iFontFamily

A font family reference for the font family whose container you want to obtain. You must pass a valid font family.

iStyle

The font style of the font family whose container you want to obtain. You must pass a valid font style.

iFontSize

The font size of the font family whose container you want to obtain. You must pass a valid font size.

oFontContainer

On output, a pointer to a file specification that specifies the name and location of the font container.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFamilyFromATSFontFamilyRef

Obtains the font family associated with an ATS font family reference. (Deprecated in Mac OS X v10.4.)

```
FMFontFamily FMGetFontFamilyFromATSFontFamilyRef (
   ATSFontFamilyRef iFamily
);
```

Parameters

iFamily
An ATS font family reference.

Return Value
The font family reference associated with the specified ATS font family reference.

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFamilyFromName

Returns the font family reference associated with a standard QuickDraw name. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyFindFromName` (page 652) instead.)

```
FMFontFamily FMGetFontFamilyFromName (
   ConstStr255Param iName
);
```

Parameters

iName
A QuickDraw font family name.

Return Value
A font family reference. The function returns `kInvalidFontFamily` if it cannot find a matching font family. See the ATS Types documentation for a description of the `FMFontFamily` data type.

Discussion
This function is a replacement for the `GetFNum` (page 1197) function. You should use the function `FMGetFontFamilyFromName` instead of the function `GetFNum` to assure your application supports font formats other than the resource fork TrueType and PostScript Type 1 fonts.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFamilyGeneration

Obtains the generation count of a font family. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetGeneration` (page 654) instead.)

```
OSStatus FMGetFontFamilyGeneration (
   FMFontFamily iFontFamily,
   FMGeneration *oGeneration
);
```

Parameters

iFontFamily
A font family reference.

oGeneration
On output, a pointer to the generation count for the font family associated with the font family reference.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFamilyInstanceFromFont

Finds the font family reference and standard QuickDraw style associated with a font. (Deprecated in Mac OS X v10.4.)

```
OSStatus FMGetFontFamilyInstanceFromFont (
   FMFont iFont,
   FMFontFamily *oFontFamily,
   FMFontStyle *oStyle
);
```

Parameters

iFont
A font reference.

oFontFamily

A pointer to a font family reference. On output, points to the font family reference associated with the specified font. You are responsible for allocating the memory for the font family reference.

oStyle

A pointer to a font style. On output, points to the font style associated with the specified font. You are responsible for allocating the memory for the font style.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
A font can be a member of more than one font family. This means if you call the function `FMGetFontFromFontFamilyInstance` (page 1187) and then call the function `FMGetFontFamilyInstanceFromFont`, you will not necessarily get the font family reference you supplied when you called `FMGetFontFromFontFamilyInstance`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFamilyName

Obtains the font family name associated with a font family reference. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetName` (page 654) instead.)

```
OSStatus FMGetFontFamilyName (
   FMFontFamily iFontFamily,
   Str255 oName
);
```

Parameters

iFontFamily

A font family reference.

oName

On output, the string contains the QuickDraw font family name. If the function does not find a name, it returns an empty string and a result code of `kFMInvalidFontFamilyErr`.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
This function is a replacement for the `GetFontName` (page 1198) function. You should use the function `FMGetFontFamilyName` instead of the function `GetFontName` to assure your application supports font formats other than the resource fork TrueType and PostScript Type 1 fonts.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFamilyResource

Obtains the font family resource for a font family. (Deprecated in Mac OS X v10.4. Use `ATSFontGetFontFamilyResource` (page 664) instead.)

```
OSStatus FMGetFontFamilyResource (
   FMFontFamily iFontFamily,
   FMFontStyle iFontStyle,
   FMFontSize iFontSize,
   ByteCount iBufferSize,
   void *ioBuffer,
   ByteCount *oSize
);
```

Parameters

iFontFamily
A value of type `FMFontFamily` that specifies the font family whose resource you want to obtain. You must pass a valid font family.

iFontStyle
A value of type `FMFontStyle` that specifies the font style of the font family whose resource you want to obtain. You must pass a valid font style.

iFontSize
A value of type `FMFontSize` that specifies the font size of the font family whose resource you want to obtain. You must pass a valid font size.

iBufferSize
The size of the buffer (`ioBuffer`).

ioBuffer
A pointer to the buffer used to store a copy of the font family resource. On input, pass `NULL` if you want to obtain only the length of the font family resource, not its contents.

oSize
On output, the actual size of the buffer.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
You should call the function `FMGetFontFamilyResource` twice. First, to get the length of the font family resource. Then after you allocate a buffer (`ioBuffer`) of the appropriate size, call the function a second time to obtain the contents of the font family resource.

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFamilyTextEncoding

Obtains the text encoding used by a font family. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetEncoding` (page 653) instead.)

```
OSStatus FMGetFontFamilyTextEncoding (
   FMFontFamily iFontFamily,
   TextEncoding *oTextEncoding
);
```

Parameters

iFontFamily

A font family reference.

oTextEncoding

On output, a pointer to the text encoding used by the font family associated with the font family reference.

Return Value

A result code. See "Font Manager Result Codes" (page 1230).

Discussion

This function is a replacement for the Script Manager function `FontToScript`. You should use the function `FMGetFontFamilyTextEncoding` instead of the function `FontToScript` to ensure your application supports font formats other than the resource fork TrueType and PostScript Type 1 fonts. Unlike the `FontToScript` function, the state of the font force flag is ignored and the script system of the font family is not mapped to zero even if the script system is disabled in the current application process.

Once you have obtained the text encoding, you can use Text Encoding Converter Manager function `RevertTextEncodingToScriptInfo` to extract the script as follows:

```
status = FMGetFontFamilyTextEncoding (myFontFamily, &myTextEncoding)

status = RevertTextEncodingToScriptInfo (myTextEncoding, &myScriptCode);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMGetFontFormat

Obtains the format identifier of a font. (Deprecated in Mac OS X v10.4.)

```
OSStatus FMGetFontFormat (
   FMFont iFont,
   FourCharCode *oFormat
);
```

Parameters

iFont

A font reference.

oFormat

On output, a pointer to a four-character-code that represents the format identifier of the font. See the discussion that follows for information on format identifiers.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
A format identifier is a four-character-code, assigned to a font by a font vendor, that identifies the format of a font. Some of the identifiers currently supported in the Mac OS are:

- `'true'` TrueType fonts use the 32-bit hexadecimal value 0x00010000.
- `'LWFN'` PostScript Type 1 fonts ("LaserWriter Font") consist of two parts: a `'FOND'` resource (contained in a font or font suitcase resource file) whose style mapping table references PostScript font data for each typeface (style), stored in separate file. The separate data files have names derived from the PostScript name of the typeface.
- `'typ1'` PostScript Type 1 fonts are housed in packages that have an `'sfnt'` format (OpenType).
- `'OTTO'` PostScript compact font format (CFF) font data is housed in a package that has an `'sfnt'` format (OpenType).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFromATSFontRef

Obtains the font object associated with an ATS font reference.

```
FMFont FMGetFontFromATSFontRef (
   ATSFontRef iFont
);
```

Parameters

iFont

An ATS font reference.

Return Value
The font reference associated with the specified ATS font reference.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontFromFontFamilyInstance

Obtains the font reference associated with a standard QuickDraw style and font family. (Deprecated in Mac OS X v10.4. Use `CTFontCreateWithQuickdrawInstance` instead.)

```
OSStatus FMGetFontFromFontFamilyInstance (
   FMFontFamily iFontFamily,
   FMFontStyle iStyle,
   FMFont *oFont,
   FMFontStyle *oIntrinsicStyle
);
```

Parameters

iFontFamily

A font family reference.

iStyle

A font style.

oFont

A pointer to a font reference. On output, points to the font reference for the specified font family and style. You are responsible for allocating the memory for the font reference.

oIntrinsicStyle

On output, a pointer to an intrinsic font style. If a font object isn't found that matches the font family reference and font style you specify, the function returns the QuickDraw style that matches most closely.

Return Value

A result code. See "Font Manager Result Codes" (page 1230). If a font reference and intrinsic style are not found, the function returns a value of `kFMInvalidFontErr`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMGetFontGeneration

Obtains the generation count of a font. (Deprecated in Mac OS X v10.4. Use `ATSFontGetGeneration` (page 665) instead.)

```
OSStatus FMGetFontGeneration (
   FMFont iFont,
   FMGeneration *oGeneration
);
```

Parameters

iFont

A font reference.

oGeneration

On output, a pointer to a value that specifies the generation count of the font.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontTable

Retrieves all or part of a data table for a font. (Deprecated in Mac OS X v10.4. Use `ATSFontGetTable` (page 668) or `CTFontCopyTable` instead.)

```
OSStatus FMGetFontTable (
   FMFont iFont,
   FourCharCode iTag,
   ByteOffset iOffset,
   ByteCount iLength,
   void *iBuffer,
   ByteCount *oActualLength
);
```

Parameters

iFont
A font reference.

iTag
A tag that identifies the data table for a font.

iOffset
An offset to font table data you want to retrieve. The offset is relative to the beginning of the data table and is zero-based.

iLength
The size of the data buffer (`ioBuffer`) you allocate.

iBuffer
A pointer to the buffer used to store a copy of the font table. On input, pass `NULL` if you want to obtain only the length of the table, not its contents.

oActualLength
On output, the actual length of the font table.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
You should call the function `FMGetFontTable` twice. First, call it to retrieve the length of the font table. Then, after you've allocated space for the `iBuffer` parameter, call the function a second time to obtain the contents of the font table.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetFontTableDirectory

Obtains the table directory for a font. (Deprecated in Mac OS X v10.4.)

```
OSStatus FMGetFontTableDirectory (
   FMFont iFont,
   ByteCount iLength,
   void *iBuffer,
   ByteCount *oActualLength
);
```

Parameters

iFont
A font reference.

iLength
The number of bytes in the buffer used to store a copy of the font table directory associated with the font.

iBuffer
A pointer to the buffer used to store a copy of the font table directory. On input, pass `NULL` if you want to obtain only the length of the table directory, not its contents.

oActualLength
On output, the length of the font table directory.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
You should call the function `FMGetFontTableDirectory` twice. First, call it to retrieve the length of the font table directory. Then, after you've allocated space for the `iBuffer` parameter, call the function a second time to obtain the contents of the table directory.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetGeneration

Retrieves the value of the generation count. (Deprecated in Mac OS X v10.5. Use `ATSGetGeneration` (page 678) instead.)

```
FMGeneration FMGetGeneration (
   void
);
```

Return Value
The generation count. See the ATS Types documentation for a description of the `FMGeneration` data type.

Discussion
Any operation that adds, deletes, or modifies one or more font families or fonts triggers an update of the global generation count. You can use this function in conjunction with the iteration functions to identify changes made to the font database.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetNextFont

Obtains the next font reference. (Deprecated in Mac OS X v10.4. Use `ATSFontIteratorNext` (page 672) instead.)

```
OSStatus FMGetNextFont (
   FMFontIterator *ioIterator,
   FMFont *oFont
);
```

Parameters

ioIterator
A pointer to a font iterator you created using the function `FMCreateFontIterator`.

oFont
A pointer to a font reference. On output, points to the next font reference obtained by the font iterator. You are responsible for allocating the memory for the font reference.

Return Value
A result code. See "Font Manager Result Codes" (page 1230). If there are no more font objects to get, the function `FMGetNextFont` returns the result code `kFMIterationCompleted`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMGetNextFontFamily

Obtains the next font family reference. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyIteratorNext` (page 657) instead.)

```
OSStatus FMGetNextFontFamily (
   FMFontFamilyIterator *ioIterator,
   FMFontFamily *oFontFamily
);
```

Parameters

ioIterator

A pointer to a font family iterator you created using the function `FMCreateFontFamilyIterator`.

oFontFamily

A pointer to a font family reference. On output, points to the font family reference obtained by the iterator. You are responsible for allocating memory for the font family reference.

Return Value

A result code. See "Font Manager Result Codes" (page 1230). If there are no more font family references to get, the function `FMGetNextFontFamily` returns the result code `kFMIterationCompleted`.

Discussion

If any changes are made to the font database while you are using the font family iterator, the iterator is invalidated and the function `FMGetNextFontFamily` returns the error `kFMIteratorScopeModified`. To remedy this error, your application must either restart or cancel the enumeration by calling the `FMResetFontFamilyIterator` (page 1193) or the `FMDisposeFontFamilyIterator` (page 1177) functions.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMGetNextFontFamilyInstance

Obtains the next instance associated with a font family reference. (Deprecated in Mac OS X v10.4.)

```
OSStatus FMGetNextFontFamilyInstance (
   FMFontFamilyInstanceIterator *ioIterator,
   FMFont *oFont,
   FMFontStyle *oStyle,
   FMFontSize *oSize
);
```

Parameters

ioIterator

A pointer to a font family instance iterator you created with the function `FMCreateFontFamilyInstanceIterator` (page 1173).

oFont

A pointer to a font reference. On output, points to the font reference obtained by the iterator. You are responsible for allocating the memory for the font reference.

oStyle

A pointer to a font style. On output, points to the font style obtained by the iterator. You are responsible for allocating the memory for the font style.

oSize

A pointer to a font size. On output, points to the font size obtained by the iterator. You are responsible for allocating the memory for the font size.

Return Value
A result code. See "Font Manager Result Codes" (page 1230). If there is no more font family information to retrieve, the function `FMGetNextFontFamlyInstance` returns the status code `kFMIterationCompleted`.

Discussion
Instances are not necessarily retrieved in the order they are listed in a Font Association Table.

If any changes are made to the font database while your application is using the font family instance iterator, the iterator is invalidated and the function `FMGetNextFontFamilyInstance` (page 1191) returns the error `kFMIteratorScopeModified`. To remedy this error, your application must either call the `FMResetFontFamilyInstanceIterator` (page 1192) or the `FMDisposeFontFamilyInstanceIterator` (page 1176) functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMResetFontFamilyInstanceIterator

Resets the a font family instance iterator to the beginning of the iteration for the specified font family. (Deprecated in Mac OS X v10.4.)

```
OSStatus FMResetFontFamilyInstanceIterator (
   FMFontFamily iFontFamily,
   FMFontFamilyInstanceIterator *ioIterator
);
```

Parameters

iFontFamily

A font family reference.

ioIterator

A pointer to a font family instance iterator you created with the function `FMCreateFontFamilyInstanceIterator`.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
Once you have created a font family instance iterator, you can reuse it by calling the function `FMResetFontFamilyInstanceIterator`. This function sets the `iFontFamily` parameter to the new font family object you specify, and repositions the iterator so it is ready to get the first font family instance when you call the function `FMGetNextFontFamilyInstance` (page 1191).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Fonts.h`

FMResetFontFamilyIterator

Resets a font family iterator to the beginning of the iteration. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyIteratorReset` (page 658) instead.)

```
OSStatus FMResetFontFamilyIterator (
   const FMFilter *iFilter,
   void *iRefCon,
   OptionBits iOptions,
   FMFontFamilyIterator *ioIterator
);
```

Parameters

iFilter

A pointer to a filter specification. Pass `NULL` if you want to access all font family objects within the scope of your iteration. Otherwise, you can use this parameter to restrict the scope of the iteration to the font families that match a generation count or criteria you specify in a custom filter function.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. If you are not using a custom filter function, pass `NULL`.

iOptions

A value that specifies the scope to which the font family iterator applies. If you want the Font Manager to apply the font family iterator only to the fonts accessible from your application use the `kFMLocalIterationScope` constant. If you want the Font Manager to apply the font family iterator to all fonts registered with the system use the constant `kFMGlobalIterationScope`.

iOIterator

A pointer to a font family iterator you created with the function `FMCreateFontFamilyIterator`. On output, the font family iterator is reset.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
Once you have created a font family iterator, you can reuse it by calling the function `FMResetFontFamilyIterator` (page 1193). This function sets the parameters to the new values you specify, and repositions the iterator so it is ready to get the first font family reference when you call the function `FMGetNextFontFamily` (page 1190).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

FMResetFontIterator

Resets a font iterator to the beginning of the iteration. (Deprecated in Mac OS X v10.4. Use `ATSFontIteratorReset` (page 674) instead.)

```
OSStatus FMResetFontIterator (
   const FMFilter *iFilter,
   void *iRefCon,
   OptionBits iOptions,
   FMFontIterator *ioIterator
);
```

Parameters

iFilter

A pointer to font filter specification. Pass `NULL` if you want to access all font objects within the scope of your iteration. Otherwise, you can use this parameter to restrict the scope of the iteration to font information that matches a technology, font container, or criteria you specify in a custom filter function. Pass the filter selector constant `kFMFontTechnologyFilterSelector` to select a font technology filter, the constant `kFMFontContainerFilterSelector` to select a font container filter, or the constant `kFMFontCallbackFilterSelector` to select a custom filter. See `FMFilterSelector` in the ATS Types Reference for more information on these constants.

iRefCon

An arbitrary 32-bit value specified by your application. If you are using a custom filter function, you can use this parameter to pass data to the custom filter function. If are not using a custom filter function, pass `NULL`.

iOptions

A value that specifies the scope to which the font iterator applies. If you want the Font Manager to apply the font iterator only to the fonts accessible from your application use the `kFMLocalIterationScope` constant. If you want the Font Manager to apply the font iterator to all fonts registered with the system use the constant `kFMGlobalIterationScope`.

ioIterator

A pointer to a font iterator you created with the function `FMCreateFontIterator` (page 1175). On output, the font iterator is not positioned on a font object, and any information about font objects that were returned previously in the font iterator is no longer available.

Return Value

A result code. See "Font Manager Result Codes" (page 1230).

Discussion

Once you have created a font iterator, you can reuse it by calling the function `FMResetFontIterator` (page 1194). This function sets the parameters to the new values you specify, and repositions the iterator so it is ready to get the first font object when you call the function `FMGetNextFont` (page 1190).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FMSwapFont

Returns a pointer to the font output structure for a specified font. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
FMOutPtr FMSwapFont (
   const FMInput *inRec
);
```

Parameters

inRec

A pointer to the font input structure for which you want to obtain font output information. A font input structure contains the font family ID, the style requested, scaling factors, and other information that specifies the characteristics of the font that is requested.

Return Value

A pointer to a font output structure (`FMOutput`). The font output structure contains a handle to the font resource for the specified input font, along with information about the font, such as the ascent, descent, and leading measurements.

Discussion

The function `FMSwapFont` is typically called by QuickDraw and other parts of the system software to access font handles. QuickDraw calls the `FMSwapFont` function every time a QuickDraw text function is used.

In most cases you don't need to call this function. If you want to call the `FMSwapFont` function to get a handle to a font resource or information about a font, you must first create a font input structure and fill it with the appropriate information. You can use the pointer returned by `FMSwapFont` to access the font output structure. You cannot assume that the font resource pointed to by the `fontHandle` field of the font output structure returned by this function is of any particular type, such as `'NFNT'` or `'sfnt'`. If you need to access specific information in the font resource, call the Resource Manager function `GetResInfo` with the handle returned in the font output structure to determine the font resource type.

The pointer to the font output structure returned by the function `FMSwapFont` points to a structure allocated in low memory by the Font Manager. The same structure is reused for each call made to `FMSwapFont`. Do not free the memory allocated for this structure.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

FontMetrics

Obtains fractional measurements for the font, size, and style specified in the current graphics port. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
void FontMetrics (
   FMetricRecPtr theMetrics
);
```

Parameters

theMetrics

A pointer to a font metrics structure. On output, the structure contains the font measurement information in fractional values.

Discussion

The `FontMetrics` function obtains measurements for the ascent, descent, leading, and width of the largest glyph in the font for the font, size, and style specified in the current graphics port.

The font metrics structure (of data type `FMetricRec`) contains a handle to the global width table, which in turn contains a handle to the associated font family resource for the current font (the font in the current graphics port). It also contains the values of four measurements for the current font.

The `FontMetrics` function is similar to the QuickDraw function `GetFontInfo` except that `FontMetrics` returns fractional values for greater accuracy in high-resolution printing. The `FontMetrics` function also does not take into account any additional widths that are added by QuickDraw when it applies styles to the glyphs in a font.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

GetAppFont

Returns the font family ID of the current application font. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
short GetAppFont (
   void
);
```

Return Value

The font family ID of the current application font. This is the font family ID that has been mapped to 1 by the system software.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

GetDefFontSize

Determines the default size of the system font. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
short GetDefFontSize (
   void
);
```

Return Value
The the default font size of the system font.

Discussion
You can determine the preferred size for either the system font or the application font of any enabled script system by calling the Script Manager function `GetScriptManagerVariable`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

GetFNum

Obtains the font family ID for a specified font family name. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyFindFromName` (page 652) instead.)

Not recommended.

```
void GetFNum (
   ConstStr255Param name,
   short *familyID
);
```

Parameters

name
The font family name.

familyID
On output, a pointer to the font family ID for the font family specified in `name`. If the font specified in the parameter `name` does not exist, the font family ID contains 0.

Carbon Porting Notes

You should use the function `FMGetFontFamilyFromName` instead of the function `GetFNum`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
Fonts.h

GetFontName

Obtains the name of a font family that has a specified family ID number. (Deprecated in Mac OS X v10.4. Use ATSFontFamilyGetName (page 654) instead.)

Not recommended.

```
void GetFontName (
   short familyID,
   Str255 name
);
```

Parameters

familyID
The font family ID.

name
On output, this parameter contains the font family name for the font family specified in familyID. If the font specified in the familyID parameter does not exist, name contains an empty string.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Fonts.h

GetOutlinePreferred

Obtains the current preference for whether outline or bitmapped fonts are returned when the Font Manager receives a font request. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
Boolean GetOutlinePreferred (
   void
);
```

Return Value
The value of the Font Manager's current preference for outline or bitmapped fonts. If GetOutlinePreferred returns TRUE, then the Font Manager will return an outline font when both an outline font and a bitmapped font are available for a particular request. If GetOutlinePreferred returns FALSE, then the Font Manager will return the bitmapped font when both types are available. See the Debugger Services documentation for a description of the Boolean data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Fonts.h

GetPreserveGlyph

Determines whether the Font Manager preserves the shapes of glyphs from outline fonts. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
Boolean GetPreserveGlyph (
   void
);
```

Return Value
A `Boolean` value that indicates whether the Font Manager preserves the shapes of glyphs from outline fonts. Your application can set the value of this variable with the `SetPreserveGlyph` function. If `GetPreserveGlyph` returns `TRUE`, the Font Manager preserves glyph shapes; if `GetPreserveGlyph` returns `FALSE`, then the Font Manager scales glyphs to fit between the ascent and descent lines for the font in use.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

GetSysFont

Obtains the font family ID of the current system font. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
short GetSysFont (
   void
);
```

Return Value
The current value of the font family ID of the current system font. This is the font family ID that has been mapped to 0 by the system software.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

IsAntiAliasedTextEnabled

Checks whether antialiased text is enabled. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
Boolean IsAntiAliasedTextEnabled (
   SInt16 *oMinFontSize
);
```

Parameters

oMinFontSize

On output, points to the minimum font size for which antialiasing is enabled.

Return Value

Returns `true` if antialiased text is enabled; `false` otherwise.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

IsOutline

Determines whether the specified scaling factors will cause the Font Manager to choose an outline font for the current graphics port. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
Boolean IsOutline (
   Point numer,
   Point denom
);
```

Parameters

numer

The numerators of the vertical and horizontal scaling factors. The `numer` parameter is of type `Point`, and contains two fields: `h` (the numerator of the ratio for horizontal scaling) and `v` (the numerator of the ratio for vertical scaling).

denom

The denominators of the vertical and horizontal scaling factors. The `denom` parameter is of type `Point`, and contains two fields: `h` (the denominator of the ratio for horizontal scaling) and `v` (the denominator of the ratio for vertical scaling).

Return Value

Returns `TRUE` if the Font Manager will choose an outline font for the current graphics port.

Discussion

The Font Manager uses the font scaling factors specified in the `numer` and `denom` parameters as well as the current preference (as set by the `SetOutlinePreferred` function) as criteria to choose which font to use.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

OutlineMetrics

Obtains font measurements for a block of text to be drawn in a specified outline font. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr OutlineMetrics (
   short byteCount,
   const void *textPtr,
   Point numer,
   Point denom,
   short *yMax,
   short *yMin,
   FixedPtr awArray,
   FixedPtr lsbArray,
   RectPtr boundsArray
);
```

Parameters

byteCount

The number of bytes in the block of text that you want measured.

textPtr

A pointer to the block of text that for which you want to obtain font measurements.

numer

The numerators of the vertical and horizontal scaling factors. The `numer` parameter is of type `Point`, and contains two fields: `h` (the numerator of the ratio for horizontal scaling) and `v` (the numerator of the ratio for vertical scaling). The Font Manager applies these scaling factors to the current font when calculating the measurements for glyphs in the block of text.

denom

The denominators of the vertical and horizontal scaling factors. The `denom` parameter is of type `Point`, and contains two fields: `h` (the denominator of the ratio for horizontal scaling) and `v` (the denominator of the ratio for vertical scaling). The Font Manager applies these scaling factors to the current font when calculating the measurements for glyphs in the block of text.

yMax

On output, a pointer to the maximum y-value for the text. Pass `NULL` in this parameter if you don't want this value returned.

yMin

On output, a pointer to the minimum y-value for the text. Pass `NULL` in this parameter if you don't want this value returned.

awArray

A pointer to an array. On output the array is filled with the advance width measurements for the glyphs being measured. These measurements are in pixels, based on the point size and font scaling factors of the current font. There is an entry in this array for each glyph that is being measured.

The `awArray` parameter is of type `FixedPtr`. The `FixedPtr` data type is a pointer to an array, and each entry in the array is of type `Fixed`, which is 4 bytes in length. Multiply `byteCount` by 4 to calculate the memory you need in bytes.

If the `FractEnable` global variable has been set to `TRUE` through the `SetFractEnable` function, the values in `awArray` have fractional character widths. If `FractEnable` has been set to `FALSE`, the Font Manager returns integer values for the advance widths, with 0 in the decimal part of the values.

lsbArray

A pointer to an array. On output the array is filled with the left-side bearing measurements for the glyphs being measured. The measurements are in pixels, based on the point size of the current font. There is an entry in this array for each glyph that is being measured.

The `lsbArray` parameter is of type `FixedPtr`. The `FixedPtr` data type is a pointer to an array, and each entry in the array is of type `Fixed`, which is 4 bytes in length. Multiply `byteCount` by 4 to calculate the memory you need in bytes.

The fractional portion of left-side bearing values are retained.

boundsArray

A pointer to an array. On output the array is filled with the bounding boxes for the glyphs being measured. Bounding boxes are the smallest rectangles that fit around the pixels of the glyph. There is an entry in this array for each glyph that is being measured.

The coordinate system used to describe the bounding boxes is in pixel units, centered at the glyph origin, and with a vertical positive direction upwards. This is the opposite of the QuickDraw vertical orientation.

The `boundsArray` parameter is of type `RectPtr`. The `RectPtr` data type is a pointer to QuickDraw's `Rect` data type, which is 8 bytes in length. Multiply `byteCount` by 8 to calculate the memory you need in bytes. Allocate the memory needed for the array and pass a pointer to the array in the `boundsArray` parameter.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Discussion
The `OutlineMetrics` function computes the maximum y-value, minimum y-value, advance widths, left-side bearings, and bounding boxes for a block of text. It uses the font, size, and style specified in the current graphics port. You can use these measurements when laying out text. You may need to adjust line spacing to accommodate exceptionally large glyphs.

The `OutlineMetrics` function works for outline fonts only and is the preferred method for measuring text that is drawn with an outline font.

When you are using `OutlineMetrics` to compute advance width values, left-side bearing values, or bounding boxes, you need to bear in mind that when a text block contains 2-byte characters, not every byte in the `awArray`, `lsbArray`, and `boundsArray` structures is used. Each of these arrays is indexed by the glyph index; thus, if you have five characters in a string, only the first five entries in each array contains a value. Call the Script Manager function `CharByte` to determine how many characters there are in the text block, and ignore the unused array entries (which occur at the end of each array).

If you don't want `OutlineMetrics` to compute one of these three values, pass `NULL` in the applicable parameter. Otherwise, allocate the amount of memory needed for the array and pass a pointer to it in this parameter.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

QDTextBounds

Obtains a rectangle that specifies the bounds of QuickDraw text. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
void QDTextBounds (
   short byteCount,
   const void *textAddr,
   Rect *bounds
);
```

Parameters

byteCount

The number of bytes in the buffer that contains the text whose bounds you want to obtain.

textAddr

A pointer to a buffer that contains the text whose bounds you want to obtain. You must allocate this buffer.

bounds

On output, points to a rectangle that specifies the bounds of QuickDraw text.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

RealFont

Determines whether a font is available or is intended for use in a specified size. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
Boolean RealFont (
   short fontNum,
   short size
);
```

Parameters

fontNum

The font family ID.

size

The font size requested.

Return Value

Returns `TRUE` if the requested size of the font is available. The function `RealFont` first checks for a bitmapped font from the specified family. If one is not available, `RealFont` checks next for an outline font. If neither kind of font is available, `RealFont` returns `FALSE`.

Discussion
If an outline font exists for the requested font family, `RealFont` normally considers the font to be available in any requested size. However, the font designer can include instructions in the font that outlines should not be used at certain point sizes, in which case the `RealFont` function considers the font unavailable and returns `FALSE`. The Font Manager determines whether the size is valid by testing the value of the smallest readable size element of the font family header table.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

SetAntiAliasedTextEnabled

Enables or disables antialiased text for an application. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSStatus SetAntiAliasedTextEnabled (
   Boolean iEnable,
   SInt16 iMinFontSize
);
```

Parameters

`iEnable`
A `Boolean` value. Pass `true` to enable antialiased text or `false` to disable it.

`iMinFontSize`
A integer of type `SInt16` that specifies the minimum font size to which antialiasing should be enabled.

Return Value
A result code. See "Font Manager Result Codes" (page 1230).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

SetFractEnable

Enables or disables fractional glyph widths. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
void SetFractEnable (
   Boolean fractEnable
);
```

Parameters

fractEnable

Specifies whether fractional widths or integer widths are to be used to determine glyph measurements. A value of TRUE indicates fractional glyph widths; a value of FALSE indicates integer glyph widths.

The SetFractEnable function assigns the value that you specify in the fractEnable parameter to the global variable FractEnable.

Discussion

The SetFractEnable function establishes whether or not the Font Manager provides fractional glyph widths to QuickDraw, which then uses them for advancing the pen during text drawing.

The Font Manager defaults to integer widths to ensure compatibility with existing applications. When fractional glyph widths are enabled, the Font Manager can determine the locations of glyphs more accurately than is possible with integer widths.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Fonts.h

SetFScaleDisable

Enables or disables the computation of font scaling factors by the Font Manager for bitmapped glyphs. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
void SetFScaleDisable (
   Boolean fscaleDisable
);
```

Parameters

fscaleDisable

Specifies whether bitmapped fonts are to be scaled. A value of TRUE indicates that font scaling is disabled; a value of FALSE indicates that font scaling is enabled.

If you set the fscaleDisable parameter to TRUE, the Font Manager disables font scaling, which means it responds to a request for a font size that is not available by computing font scaling factors of 1/1 and returning a smaller, unscaled bitmapped font with the widths of the requested size. If you set the fscaleDisable parameter to FALSE, the Font Manager computes scaling factors for bitmapped fonts.

Discussion

QuickDraw performs the actual scaling of glyph bitmaps for bitmapped fonts by using the font scaling factors computed and returned by the Font Manager.

When font scaling is enabled, the Font Manager can scale a bitmapped glyph that is present in the System file to imitate the appearance of a bitmapped glyph in another point size that is not present. By default, the Font Manager scales fonts to ensure compatibility with existing applications.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

SetOutlinePreferred

Sets the preference for whether to use bitmapped or outline fonts when both kinds of fonts are available. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
void SetOutlinePreferred (
   Boolean outlinePreferred
);
```

Parameters

`outlinePreferred`

Specifies whether the Font Manager chooses an outline font or a bitmapped font when both are available to fill a font request. A value of `TRUE` indicates an outline font; a value of `FALSE` indicates a bitmapped font.

If you want the Font Manager to choose outline fonts over any bitmapped font counterparts, set the `outlinePreferred` parameter to `TRUE`. If you want it to choose bitmapped fonts, set the `outlinePreferred` parameter to `FALSE`.

Discussion
If an outline font and a bitmapped font are both available for a font request, the default behavior for the Font Manager is to choose the bitmapped font, in order to maintain compatibility with documents that were created on computer systems on which outline fonts were not available. The `SetOutlinePreferred` function sets the Font Manager's current preference for either bitmapped or outline fonts when both are available.

If only outline fonts are available, the Font Manager chooses them regardless of the value of the `outlinePreferred` parameter. If only bitmapped fonts are available, they are chosen. The Font Manager chooses bitmapped versus outline fonts on a size basis, before it takes stylistic variations into account, which can lead to unexpected results.

The preference you set is valid only during the current session with your application. The `outlinePreferred` parameter does not set a global variable.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Fonts.h`

SetPreserveGlyph

Temporarily changes the default behavior of the Font Manager, so that it does not scale oversized glyphs. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
void SetPreserveGlyph (
   Boolean preserveGlyph
);
```

Parameters

preserveGlyph

Specifies whether or not glyphs from an outline font are scaled to fit between the ascent and descent lines. If you set the value of the `preserveGlyph` parameter to `TRUE`, the measurements of all glyphs are preserved, which means that your application may have to alter the leading between lines in a document if some of the glyphs extend beyond the ascent or descent lines. If you set the value of the `preserveGlyph` parameter to `FALSE`, all glyphs are scaled to fit between the ascent and descent lines.

Discussion

The `SetPreserveGlyph` function establishes how the Font Manager treats glyphs that do not fit between the ascent and descent lines for the current font. The default behavior for the Font Manager is to scale a glyph from an outline font so that it fits between the ascent and descent lines; however, this alters the appearance of the glyph.

You can determine the current behavior of the Font Manager in this regard by calling the `GetPreserveGlyph` function. To ensure that documents have the same appearance whenever they are opened, you need to call `GetPreserveGlyph` and save the value that it returns with your documents and restore it each time a document is displayed by your application.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Fonts.h`

Data Types

Font and Font Family Data Structures

FMFontContainer

Represents a font container.

```
typedef UInt32 FMFontContainer;
```

FMFontInstance

Contains information for a font instance.

```
struct FMFontInstance {
    FMFont font;
    UInt32 fontInstanceIndex;
};
```

Fields

`font`
A font reference.

`fontInstanceIndex`
The index associated with the font.

FMFontSpecification

Contains a font family and style.

```
struct FMFontSpecification {
    FMFontFamily fontFamily;
    SInt16 style;
};
```

Fields

`fontFamily`
A font family reference.

`style`
A font style.

FontFamilyID

Represents the ID of a font family.

```
typedef FMFontFamily FontFamilyID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

Font Input and Output Structures

FMInput

Contains information about a specific font.

```
struct FMInput {
    SInt16 family;
    SInt16 size;
    Style face;
    Boolean needBits;
    SInt16 device;
    Point numer;
    Point denom;
    float x;
    float y;
    float width;
    float height;
    CGPoint origin;
    CGSize size;
};
```

Fields

`family`
The font family ID.

`size`
The point size of the font.

`face`
The font style. The defined QuickDraw styles are bold, italic, underline, outline, shadow, condense, and extend.

`needBits`
Indicates whether QuickDraw draws the glyphs. If QuickDraw does not draw the glyphs, as is the case for measurement functions such as `MeasureText`, then the glyph bitmaps do not have to be read or constructed. If QuickDraw draws the glyphs and the font is contained in a bitmapped font resource, all of the information describing the font, including the bit image, is read into memory.

`device`
This is no longer used. The high-order byte contains the printer driver reference number as defined in the old Printing Manager. The low-order byte is reserved.

`numer`
The numerators of the vertical and horizontal scaling factors. The `numer` field is of type `Point` and contains two fields: `h` (the numerator of the ratio for horizontal scaling) and `v` (the numerator of the ratio for vertical scaling).

`denom`
The denominators of the vertical and horizontal scaling factors. The `denom` field is of type `Point` and contains two fields: `h` (the denominator of the ratio for horizontal scaling) and `v` (the denominator of the ratio for vertical scaling).

Discussion

The font input structure, of data type `FMInput`, is used by QuickDraw when it requests a font from the Font Manager. You can also use this data type when you request a font with the `FMSwapFont` (page 1195) function.

FMOutPtr

Defines a reference to a font output structure.

```
typedef FMOutputPtr FMOutPtr;
```

Discussion
See FMOutput (page 1210).

Availability
Available in Mac OS X v10.0 and later.

Declared In
Fonts.h

FMOutput

Contains a handle to a font resource and measurement and display information about a specific font.

```
struct FMOutput {
    SInt16 errNum;
    Handle fontHandle;
    UInt8 boldPixels;
    UInt8 italicPixels;
    UInt8 ulOffset;
    UInt8 ulShadow;
    UInt8 ulThick;
    UInt8 shadowPixels;
    SInt8 extra;
    UInt8 ascent;
    UInt8 descent;
    UInt8 widMax;
    SInt8 leading;
    SInt8 curStyle;
    Point numer;
    Point denom;
};
typedef struct FMOutput FMOutput;
```

Fields

`errNum`
Reserved for use by Apple Computer, Inc.

`fontHandle`
A handle to the font resource. The font resource can be for either a bitmapped font or outline font resource.

`boldPixels`
A value used by QuickDraw to modify how it applies the bold style on the screen and on raster printers. Other display devices may handle styles differently.

`italicPixels`
A value used by QuickDraw to modify how it applies the italic style on the screen and on raster printers. Other display devices may handle styles differently.

`ulOffset`
A value used by QuickDraw to modify how it applies the underline style on the screen and on raster printers. Other display devices may handle styles differently.

`ulShadow`
A value used by QuickDraw to modify how it applies the underline shadow style on the screen and on raster printers. Other display devices may handle styles differently.

`ulThick`
: A value used by QuickDraw to modify how it applies the thickness of the underline style on the screen and on raster printers. Other display devices may handle styles differently.

`shadowPixels`
: A value used by QuickDraw to modify how it applies the shadow style on the screen and on raster printers. Other display devices may handle styles differently.

`extra`
: The number of pixels by which the styles have widened each glyph.

`ascent`
: The ascent measurement of the font. Any algorithmic styles or stretching that may be applied to the font are not taken into account for this value.

`descent`
: The descent measurement of the font. Any algorithmic styles or stretching that may be applied to the font are not taken into account for this value.

`widMax`
: The maximum width of the font. Any algorithmic styles or stretching that may be applied to the font are not taken into account for this value.

`leading`
: The leading assigned to the font. Any algorithmic styles or stretching that may be applied to the font are not taken into account for this value.

`curStyle`
: The actual style being made available for QuickDraw's text drawing, as opposed to the requested style.

`numer`
: The numerators of the vertical and horizontal scaling factors. The `numer` parameter is of type `Point`, and contains two fields: `h` (the numerator of the ratio for horizontal scaling) and `v` (the numerator of the ratio for vertical scaling).

`denom`
: The denominators of the vertical and horizontal scaling factors. The `demon` parameter is of type `Point`, and contains two fields: `h` (the denominator of the ratio for horizontal scaling) and `v` (the denominator of the ratio for vertical scaling).

Discussion

The font output structure, of data type `FMOutput`, contains a handle to a font and information about font measurements. It is filled in by the Font Manager upon responding to a font request. You can request a font using the `FMSwapFont` (page 1195) function.

The `bold`, `italic`, `ulOffset`, `ulShadow`, `ulThick`, and `shadow` values are all used to communicate to QuickDraw how to modify the way it renders each stylistic variation. Each byte value is taken from the font characterization table of the printer driver and is used by QuickDraw when it draws to a screen or raster printer.

The `ascent`, `descent`, `widMax`, and `leading` values can all be different in this structure than the corresponding values in the `FontInfo` structure that is produced by the `GetFontInfo` function in QuickDraw. This is because `GetFontInfo` takes into account any algorithmic styles or stretching that QuickDraw performs, while the Font Manager functions do not.

The `numer` and `denom` values are used to designate how font scaling is to be done. The values for these fields in the font output structure can be different than the values specified in the font input structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

FMOutputPtr

Defines a pointer to a font output structure.

```
typedef FMOutput* FMOutputPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

Font Measurements

FMetricRec

Contains font measurements.

```
struct FMetricRec {
    Fixed ascent;
    Fixed descent;
    Fixed leading;
    Fixed widMax;
    Handle wTabHandle;
};
typedef struct FMetricRec FMetricRec;
```

Fields

`ascent`
: The measurement, in pixels, from the baseline to the ascent line of the font. You can determine the line height, in pixels, by adding the values of the `ascent`, `descent`, and `leading` fields of the font metrics structure.

`descent`
: The measurement, in pixels, from the baseline to the descent line of the font.

`leading`
: The measurement, in pixels, from the descent line to the ascent line below it.

`widMax`
: The width, in pixels, of the largest glyph in the font.

`wTabHandle`
: A handle to the global font width table.

Discussion
The font metrics structure (of data type `FMetricRec`) contains a handle to the global width table, which in turn contains a handle to the associated font family resource for the current font (the font in the current graphics port). It also contains the values of four measurements for the current font.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

FMetricRecHandle

Defines a handle to a font metrics structure.

```
typedef FMetricRecPtr* FMetricRecHandle;
```

Discussion
See FMetricRec (page 1212).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

FMetricRecPtr

Defines a pointer to a font metrics structure.

```
typedef FMetricRec* FMetricRecPtr;
```

Discussion
See FMetricRec (page 1212).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

FontPointSize

Represents the point size of a font.

```
typedef FMFontSize FontPointSize;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

Deprecated Data Types

The following data structures referenced by the low memory global variables of the Font Manager are deprecated in Mac OS X and CarbonLib 1.1. The low memory global variables are not shared between processes and may result in inconsistencies compared to previous releases of the system software. Changes made to the information contained in the low memory global variables, including any indirectly reference width tables, font family records, and font records, are not reflected in the global state of the Font Manager and may only be accessed through the font access and data management function of the Font Manager or ATS.

AsscEntry

Contains the size and style for a specific font.

```
struct AsscEntry {
    SInt16 fontSize;
    SInt16 fontStyle;
    SInt16 fontID;
};
```

Fields

`fontSize`
A font point size.

`fontStyle`
A font style.

`fontID`
A font Resource ID.

Discussion
The font association entry structure is used in `FontAssoc` (page 1216).

FamRec

Contains format information for a font family resource.

```
struct FamRec {
    SInt16 ffFlags;
    SInt16 ffFamID;
    SInt16 ffFirstChar;
    SInt16 ffLastChar;
    SInt16 ffAscent;
    SInt16 ffDescent;
    SInt16 ffLeading;
    SInt16 ffWidMax;
    SInt32 ffWTabOff;
    SInt32 ffKernOff;
    SInt32 ffStylOff;
    SInt16 ffProperty[9];
    SInt16 ffIntl[2];
    SInt16 ffVersion;
};
```

Fields

`ffFlags`
: Flags for family.

`ffFamID`
: Family ID number.

`ffFirstChar`
: ASCII code of first character.

`ffLastChar`
: ASCII code of last character.

`ffAscent`
: Maximum ascent for 1-point font.

`ffDescent`
: Maximum descent for 1-point font.

`ffLeading`
: Maximum leading for 1-point font.

`ffWidMax`
: Maximum glyph width for 1-point font.

`ffWTabOff`
: Offset to family glyph-width table.

`ffKernOff`
: Offset to kerning table.

`ffStylOff`
: Offset to style-mapping table.

`ffProperty`
: Style properties info.

`ffIntl`
: For international use.

`ffVersion`
: Version number.

Discussion
The font family structure, of data type `FamRec`, describes the format of the font family (`'FOND'`) resource. It is shown here as a guide to the format of the resource. The font family structure is not used directly by any Font Manager functions.

FontAssoc

Contains the number of entries in a font association table.

```
struct FontAssoc {
    SInt16 numAssoc;
};
```

Fields
`numAssoc`
Number of entries - 1.

Discussion
Each entry in the font association table is a font association entry structure, of data type `AsscEntry` (page 1214).

The font association table structure, which is part of the font family resource, maps a point size and style to a specific font that is part of the family. The table structure, of data type `FontAssoc`, consists of a count of the entries in the table and is followed by the entry structures.

FontRec

Contains information for a format of `'NFNT'` and, likewise, the `'FONT'` resource

```
struct FontRec {
    SInt16 fontType;
    SInt16 firstChar;
    SInt16 lastChar;
    SInt16 widMax;
    SInt16 kernMax;
    SInt16 nDescent;
    SInt16 fRectWidth;
    SInt16 fRectHeight;
    UInt16 owTLoc;
    SInt16 ascent;
    SInt16 descent;
    SInt16 leading;
    SInt16 rowWords;
};
typedef struct FontRec FontRec;
```

Fields
`fontType`
Font type.
`firstChar`
Character code of first glyph.
`lastChar`
Character code of last glyph.

`widMax`
Maximum glyph width.

`kernMax`
Negative of maximum glyph kern.

`nDescent`
Negative of descent.

`fRectWidth`
Width of font rectangle.

`fRectHeight`
Height of font rectangle.

`owTLoc`
Location of width/offset table.

`ascent`
Ascent.

`descent`
Descent.

`leading`
Leading.

`rowWords`
Row width of bit image / 2.

Discussion
The font structure, of data type `FontRec`, describes the format of `'NFNT'` and, likewise, the `'FONT'` resource. It is shown here as a guide to the format of the resource. The font structure is not used directly by any Font Manager functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

FontRecHdl

Defines a handle to a font record.

```
typedef FontRecPtr* FontRecHdl;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

FontRecPtr

Defines a pointer to a font record.

```
typedef FontRec* FontRecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

KernEntry

Contains kerning information for a specific stylistic variation of the font family.

```
struct KernEntry {
    SInt16 kernStyle;
    SInt16 kernLength;
};
```

Fields

`kernStyle`
Length of this entry.

`kernLength`
Style to which this entry applies.

KernPair

Specifies a kerning value for a pair of glyphs.

```
struct KernPair {
    char kernFirst;
    char kernSecond;
    SInt16 kernWidth;
};
```

Fields

`kernFirst`
ASCII character code of the first character of a kerned pair.

`kernSecond`
ASCII character code of the second character of a kerned pair.

`kernWidth`
Kerning value in 1pt fixed format.

KernTable

Contains the number of entries in a kerning table.

```
struct KernTable {
    SInt16 numKerns;
};
```

Fields

`numKerns`
Number of subtable entries.

Discussion

The font family kerning table structure, which is part of the font family resource, contains a number of kerning subtable entries, with different subtables for different stylistic variations. The table structure, of data type `KernTable`, consists of a count of the entries in the table and is followed by the entry structures.

NameTable

Contains the base name and suffixes for a font family.

```
struct NameTable {
    SInt16 stringCount;
    Str255 baseFontName;
};
```

Fields

`stringCount`
The number of entries in the name table.

`baseFontName`
A string that specifies the base name and suffixes for a font family name.

StyleTable

Contains font style information

```
struct StyleTable {
    SInt16 fontClass;
    SInt32 offset;
    SInt32 reserved;
    char indexes[48];
};
```

Fields

`fontClass`
The font class of this table.

`offset`
Offset to glyph-encoding subtable.

`reserved`
Reserved.

`indexes`
Indexes into the font suffix name table. The font suffix name subtable structure, of data type `NameTable` (page 1219), contains the base name and suffixes for a font family.

Discussion
The style-mapping table structure, which is part of the font family resource, provides information that is used by printer drivers to implement font styles. Each font family can have its own character encoding and its own set of font suffix names for style designations. Each style of a font has its own name, typically created by adding a style suffix to the base name of the font.

WidEntry

Specifies a style for a glyph width.

```
struct WidEntry {
    SInt16 widStyle;
};
```

Fields
`widStyle`
The style to which the entry applies.

WidTable

Specifies the number of entries in a font family glyph-width table.

```
struct WidTable {
    SInt16 numWidths;
};
```

Fields
`numWidths`
The number of entries minus one.

Discussion
The font family glyph-width table structure, which is part of the font family resource, is used to specify glyph widths for the font family on a per-style basis.

WidthTable

Contains the widths of all the glyphs of a specific font.

```
struct WidthTable {
    Fixed tabData[256];
    Handle tabFont;
    SInt32 sExtra;
    SInt32 style;
    SInt16 fID;
    SInt16 fSize;
    SInt16 face;
    SInt16 device;
    Point inNumer;
    Point inDenom;
    SInt16 aFID;
    Handle fHand;
    Boolean usedFam;
    UInt8 aFace;
    SInt16 vOutput;
    SInt16 hOutput;
    SInt16 vFactor;
    SInt16 hFactor;
    SInt16 aSize;
    SInt16 tabSize;
};
typedef struct WidthTable WidthTable;
```

Fields

`tabData`
: The widths for the glyphs in the font, in standard 32-bit fixed-point format. If a glyph is missing in the font, its entry contains the width of the missing-character glyph.

`tabFont`
: A handle to the font resource used to build this table.

`sExtra`
: The average number of pixels by which QuickDraw widens each space in a line of text.

`style`
: The average number of pixels by which QuickDraw widens a line of text after applying a style.

`fID`
: The font family ID of the font represented by this table. This is the ID that was used in the request to build the table. It may be different from the ID of the font family that was used, which is indicated by the `aFID` field.

`fSize`
: The point size that was originally requested for the font represented by this table. The actual size used is specified in the `aSize` field.

`face`
: The font style that was originally requested for the font represented by this table. The actual style used is specified in the `aFace` field.

`device`
: The device ID of the device on which these widths may be used.

`inNumer`
: The numerators of the vertical and horizontal scaling factors. The `numer` parameter is of type `Point`, and contains two fields: `h` (the numerator of the ratio for horizontal scaling) and `v` (the numerator of the ratio for vertical scaling).

`inDenom`
: The denominators of the vertical and horizontal scaling factors. The `denom` parameter is of type `Point`, and contains two fields: `h` (the denominator of the ratio for horizontal scaling) and `v` (the denominator of the ratio for vertical scaling).

`aFID`
: The font family ID of the font family actually used to build this table. If the Font Manager could not find the font requested, this value may be different from the value of the `fID` field.

`fHand`
: The handle to the font family resource used to build this table.

`usedFam`
: Set to `TRUE` if the fixed-point family glyph widths were used rather than integer glyph widths.

`aFace`
: The font style of the font whose widths are contained in this table.

`vOutput`
: The factor by which glyphs are to be expanded vertically in the current graphics port. This is a 16-bit fixed-point number, with the integer part in the high-order byte and a fractional part in the low-order byte.

`hOutput`
: The factor by which glyphs are to be expanded horizontally in the current graphics port. This is a 16-bit fixed-point number, with the integer part in the high-order byte and a fractional part in the low-order byte.

`vFactor`
: The factor by which widths of the chosen font, after a style has been applied, have been increased vertically in the current graphics port. This is a 16-bit fixed-point number, with the integer part in the high-order byte and a fractional part in the low-order byte. The value of the `vFactor` field is not used by the Font Manager.

`hFactor`
: The factor by which widths of the chosen font, after a style has been applied, have been increased horizontally in the current graphics port. This is a 16-bit fixed-point number, with the integer part in the high-order byte and a fractional part in the low-order byte.

`aSize`
: The size of the font actually used to build this table. Both the point size and the font used to build this table may be different from the requested point size and font. If font scaling is disabled, the Font Manager may use a size different from the size requested and add more or less space to approximate the appearance of the font requested.

`tabSize`
: The total size of the global width table.

Discussion

The global width table structure, of data type `WidthTable`, contains the widths of all the glyphs of one font. The font family, point size, and style of this font are specified in this table. Your application should use the widths found in the global width table for placement of glyphs and words both on the screen and on the printed page. You can use the `FontMetrics` (page 1195) function to get a handle to the global width table. However, you should not assume that the table is the same size as shown in the structure declaration; it may be larger because of some private system-specific information that is attached to it.

Multiplying the values of the `hOutput` and `vOutput` fields by the values of the `hFactor` and `vFactor` fields, respectively, gives the font scaling. (Because the value of the `vFactor` field is ignored, the Font Manager multiplies the value of the `vOutput` field by 1.) The product of the value of the `hOutput` field and an entry in the global width table is the scaled width for that glyph.

The Font Manager gathers data for the global width table from one of three data structures:

1. The Font Manager looks in the font resource for a table that stores fractional glyph widths. For bitmapped fonts, the Font Manager uses the glyph-width table of the bitmapped font resource. For outline fonts, the Font Manager uses the advance width and left-side bearing values in the horizontal metrics table of the outline font. In both cases, the values are stored in 16-bit fixed format, with the integer part in the high-order byte and the fractional part in the low-order byte.
2. If there is no glyph-width table in the font resource, the Font Manager looks for the font family's glyph-width table in the font family resource, which contains fractional widths for a hypothetical 1-point font. The Font Manager calculates the actual values by multiplying these widths by the requested font size.
3. If there is no glyph-width table in the font family resource, and if the font is contained in a bitmapped font resource, the Font Manager derives the glyph widths from the integer widths contained in the glyph-width table of the bitmapped font resource. There is no corresponding table for the outline font resource.

Your application should obtain glyph widths either from the global width table or from the QuickDraw function `MeasureText`. The `MeasureText` function works only with text to be displayed on the screen, not with text to be printed. You can get the individual widths of glyphs of an outline font using the `OutlineMetrics` function. The `FontMetrics` function returns only the width of the largest glyph in a font contained in a bitmapped font resource.

Do not use the values from the global width table if your application is running on a computer on which non-Roman script systems are installed. You can check to see if a non-Roman script system is present by calling the `GetScriptManagerVariable` function with a selector of `smEnabled`; if the function returns a value greater than 0, at least one non-Roman script system is present and you need to call `MeasureText` to measure text that is displayed on the screen. Measuring text from a non-Roman script system for printing is handled by the printer driver.

For more information about the `MeasureText` function, see the documentation on "QuickDraw Text". See also the `FontMetrics` (page 1195) and `OutlineMetrics` (page 1201) functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

WidthTableHdl

Defines a handle to a glyph width table.

```
typedef WidthTablePtr* WidthTableHdl;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

WidthTablePtr

Defines a pointer to a glyph width table.

```
typedef WidthTable* WidthTablePtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Fonts.h`

Constants

Activation Contexts

Specify the scope of available fonts.

```
enum{
kFMDefaultActivationContext = kFMDefaultOptions,
kFMGlobalActivationContext = 0x00000001,
kFMLocalActivationContext = kFMDefaultActivationContext
}
```

Constants

`kFMDefaultActivationContext`
Specifies to use the default scope, which is local.
Available in Mac OS X v10.1 and later.
Declared in `Fonts.h`.

`kFMGlobalActivationContext`
Specifies the scope is global; fonts are available to all applications.
Available in Mac OS X v10.1 and later.
Declared in `Fonts.h`.

`kFMLocalActivationContext`
Specifies the scope is local; fonts are available only to the application.
Available in Mac OS X v10.1 and later.
Declared in `Fonts.h`.

Default Options

Specify the scope of fonts for an application.

```
enum {
    kFMDefaultOptions = kNilOptions
};
```

Constants

`kFMDefaultOptions`

Restricts the scope only to the fonts accessible to your application. This flag is also used when Apple has not yet defined options for a function that has an options parameter. When no options are defined yet, you can use `kFMDefaultOptions` as a neutral value to assure future compatibility.

Available in Mac OS X v10.0 and later.

Declared in `Fonts.h`.

Font ID Constants

Specify a font. These constants are deprecated.

```
enum {
    kFontIDNewYork = 2,
    kFontIDGeneva = 3,
    kFontIDMonaco = 4,
    kFontIDVenice = 5,
    kFontIDLondon = 6,
    kFontIDAthens = 7,
    kFontIDSanFrancisco = 8,
    kFontIDToronto = 9,
    kFontIDCairo = 11,
    kFontIDLosAngeles = 12,
    kFontIDTimes = 20,
    kFontIDHelvetica = 21,
    kFontIDCourier = 22,
    kFontIDSymbol = 23,
    kFontIDMobile = 24
};
```

Font Constants

Specify a font. These constants are deprecated.

```
enum {
    newYork = 2,
    geneva = 3,
    monaco = 4,
    venice = 5,
    london = 6,
    athens = 7,
    sanFran = 8,
    toronto = 9,
    cairo = 11,
    losAngeles = 12,
    times = 20,
    helvetica = 21,
    courier = 22,
    symbol = 23,
    mobile = 24
};
```

Discussion
You should use the functions `GetFNum` or `FMGetFontFamilyFromName` to find a font family from a standard QuickDraw name.

Global Scope Option

```
enum {
kFMUseGlobalScopeOption
};
```

Discussion
Use the constant `kFMGlobalIterationScope` instead; `kFMUseGlobalScopeOption` is deprecated.

Height and Width Constants

Specify proportional or fixed font heights and widths.

```
enum {
    propFont = 36864,
    prpFntH = 36865,
    prpFntW = 36866,
    prpFntHW = 36867,
    fixedFont = 45056,
    fxdFntH = 45057,
    fxdFntW = 45058,
    fxdFntHW = 45059,
    fontWid = 44208
};
```

Constants
`propFont`
Specifies a proportional font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`prpFntH`
Specifies a proportional height font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`prpFntW`
Specifies a proportional width font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`prpFntHW`
Specifies a proportional width and height font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`fixedFont`
Specifies a fixed font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`fxdFntH`
Specifies a fixed height font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`fxdFntW`
Specifies a fixed width font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`fxdFntHW`
Specifies a fixed height and width font.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`fontWid`
Specifies a font width.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

Iteration Scopes

Specify a scope over which to iterate.

```
enum{
kFMDefaultIterationScope = kFMDefaultOptions,
kFMGlobalIterationScope = 0x00000001,
kFMLocalIterationScope = kFMDefaultIterationScope
}
```

Constants

`kFMDefaultIterationScope`
Specifies to use the default.
Available in Mac OS X v10.1 and later.
Declared in `Fonts.h`.

`kFMGlobalIterationScope`
Specifies the scope is global, iterate over all applications.
Available in Mac OS X v10.1 and later.
Declared in `Fonts.h`.

`kFMLocalIterationScope`
Specifies the scope is local, restrict the iteration to the application.
Available in Mac OS X v10.1 and later.
Declared in `Fonts.h`.

Marking Character Constants

Specify a character to use for an active menu or submenu item.

```
enum {
    commandMark = 17,
    checkMark = 18,
    diamondMark = 19,
    appleMark = 20
};
```

Constants

`commandMark`
Specifies to use a command mark next to an active menu or submenu item.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`checkMark`
Specifies to use a check mark next to an active menu or submenu item.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`diamondMark`
Specifies to use a diamond mark next to an active menu or submenu item.
Available in Mac OS X v10.0 and later.
Declared in `Fonts.h`.

`appleMark`
Specifies to use an Apple character next to an active menu or submenu item.

Available in Mac OS X v10.0 and later.

Declared in `Fonts.h`.

Discussion
You can pass these constants in the `markChar` parameter of the Menu Manager function `GetItemMark` and the marking character field of the menu resource (of type `'MENU'`) and return these constants in the `markChar` parameter of the Menu Manager function `SetItemMark` to specify the mark of a specific menu item or the menu ID of the submenu associated with the menu item.

QuickTime User Interface Default Font

Defines the default font for the QuickTime user interface.

```
enum {
    kPlatformDefaultGuiFontID = applFont;
};
```

Constants
`kPlatformDefaultGuiFontID`
Specifies that the default font ID for the graphical user interface in QuickTime 3.0 should be the application font ID.

Available in Mac OS X v10.0 and later.

Declared in `Fonts.h`.

Discussion
This constant is used in QuickTime 3.0.

System and Application Fonts

Specify the current system and application fonts.

```
enum {
    systemFont = 0,
    applFont = 1
};
```

Constants
`systemFont`
Specifies the current System font.

Available in Mac OS X v10.0 and later.

Declared in `Fonts.h`.

`applFont`
Specifies the current application font.

Available in Mac OS X v10.0 and later.

Declared in `Fonts.h`.

Result Codes

The most common result codes returned by Font Manager are listed below.

Result Code	Value	Description
`kFMIterationCompleted`	-980	Iteration successfully completed. Available in Mac OS X v10.0 and later.
`kFMInvalidFontFamilyErr`	-981	The specified font family is invalid. Available in Mac OS X v10.0 and later.
`kFMInvalidFontErr`	-982	The specified font is invalid. Available in Mac OS X v10.0 and later.
`kFMIterationScopeModifiedErr`	-983	Iteration scope modified. Available in Mac OS X v10.0 and later.
`kFMTableAccessErr`	-984	The table specified is invalid or doesn't exist.
`kFMFontContainerAccessErr`	-985	Not able to access font container. Available in Mac OS X v10.0 and later.

Icon Services and Utilities Reference

Framework: Carbon/Carbon.h

Declared in Icons.h

Overview

The Icon Utilities allow your application (and system software) to manipulate and draw icons of any standard resource type in windows and if necessary in menus or dialog boxes. You need to use these routines only if you wish to draw icons in your application's windows or to draw icons whose resource types are not recognized by the Menu Manager and Dialog Manager in menus and dialog boxes.

To display an icon most effectively at a variety of sizes and bit depths, you should provide an icon family. You can then draw the appropriate member of the family for a given size and bit depth either by passing the family's resource ID to an Icon Utilities routine or by reading the family's icon resources into memory as an icon suite and passing the suite's handle to Icon Utilities routines.

Icon Services provides icon data to multiple Mac OS clients, including the Finder, extensions and applications. Using Icon Services to obtain icon data means you can provide efficient icon caching and release memory when you don't need icon data any longer. Icon Services provides the appropriate icon for any file object (file, folder, or volume), as well as other commonly used icons such as caution, note, or help icons in alert boxes, for example. The icons provided by Icon Services support a much larger palette of colors: up to 24 bits per pixel and an eight-bit mask. Icons are Appearance-compliant and appropriate to the active theme.

Functions by Task

Converting an Icon Mask to a Region

`IconIDToRgn` (page 1260) Deprecated in Mac OS X v10.5
: Converts the icon mask in an icon family to a region. (Deprecated. Use Icon Services instead.)

`IconMethodToRgn` (page 1261) Deprecated in Mac OS X v10.5
: Converts, to a region, the mask for an icon that `IconMethodToRgn` obtains with the aid of your icon getter callback function. (Deprecated. Use Icon Services instead.)

`IconSuiteToRgn` (page 1267) Deprecated in Mac OS X v10.5
: Converts the icon mask in an icon suite to a region. (Deprecated. Use Icon Services instead.)

Creating an Icon Suite

`AddIconToSuite` (page 1238) Deprecated in Mac OS X v10.5
Adds an icon to an icon suite. (Deprecated. Use Icon Services instead.)

`GetIconSuite` (page 1257) Deprecated in Mac OS X v10.5
Creates an icon suite in memory that contains handles to a specified icon family's resources. (Deprecated. Use Icon Services instead.)

`NewIconSuite` (page 1273) Deprecated in Mac OS X v10.5
Gets a handle to an empty icon suite. (Deprecated. Use Icon Services instead.)

Determining Whether a Point Is Within an Icon

`PtInIconID` (page 1285) Deprecated in Mac OS X v10.5
Determines whether a specified point is within an icon. (Deprecated. Use Icon Services instead.)

`PtInIconMethod` (page 1285) Deprecated in Mac OS X v10.5
Determines whether a specified point is within an icon obtained with the aid of your icon getter callback function. (Deprecated. Use Icon Services instead.)

`PtInIconSuite` (page 1288) Deprecated in Mac OS X v10.5
Determines whether a specified point is within an icon. (Deprecated. Use Icon Services instead.)

Determining Whether a Rectangle Intersects an Icon

`RectInIconID` (page 1289)
Hit-tests a rectangle against the appropriate icon mask from an icon family for a specified destination rectangle and alignment. (Deprecated. Use Icon Services instead.)

`RectInIconMethod` (page 1290)
Hit-tests a rectangle against an icon obtained by your icon getter callback function for a specified destination rectangle and alignment. (Deprecated. Use Icon Services instead.)

`RectInIconSuite` (page 1292)
Hit-tests a rectangle against the appropriate icon mask from an icon suite for a specified destination rectangle and alignment. (Deprecated. Use Icon Services instead.)

Disposing of Icon Suites

`DisposeIconSuite` (page 1241) Deprecated in Mac OS X v10.5
Releases the memory occupied by an icon suite. (Deprecated. Use Icon Services instead.)

Disposing of Icons

`DisposeCIcon` (page 1240) Deprecated in Mac OS X v10.5
Releases the memory occupied by a color icon structure. (Deprecated. Use Icon Services instead.)

Drawing Icons From an Icon Suite

`PlotIconSuite` (page 1282) Deprecated in Mac OS X v10.5
: Draws the icon described by an icon suite using the most appropriate icon in the suite for the current bit depth of the display device and the rectangle in which the icon is to be drawn. (Deprecated. Use Icon Services instead.)

Drawing Icons From Resources

`PlotCIcon` (page 1275) Deprecated in Mac OS X v10.5
: Draws a color icon of resource type `'cicn'` to which you have a handle. (Deprecated. Use Icon Services instead.)

`PlotCIconHandle` (page 1276) Deprecated in Mac OS X v10.5
: Draws an icon of resource type `'cicn'` to which you have a handle. (Deprecated. Use Icon Services instead.)

`PlotIcon` (page 1277) Deprecated in Mac OS X v10.5
: Draws an icon of resource type `'ICON'` to which you have a handle. (Deprecated. Use Icon Services instead.)

`PlotIconHandle` (page 1278) Deprecated in Mac OS X v10.5
: Draws an icon of resource type `'ICON'` or `'ICN#'` to which you have a handle. (Deprecated. Use Icon Services instead.)

`PlotIconID` (page 1279) Deprecated in Mac OS X v10.5
: Draws the icon described by an icon family. (Deprecated. Use Icon Services instead.)

`PlotIconMethod` (page 1280) Deprecated in Mac OS X v10.5
: Draws an icon obtained with the aid of your icon getter callback function. (Deprecated. Use Icon Services instead.)

`PlotSICNHandle` (page 1284) Deprecated in Mac OS X v10.5
: Draws a small icon of resource type `'SICN'` to which you have a handle. (Deprecated. Use Icon Services instead.)

Enabling and Disabling Custom Icons

`GetCustomIconsEnabled` (page 1245)
: Determines whether custom icons are enabled or disabled on a specified volume.

`SetCustomIconsEnabled` (page 1297)
: Enables or disables custom icons on a specified volume.

Flushing IconRef Data

`FlushIconRefs` (page 1242) Deprecated in Mac OS X v10.3
: Reclaims memory used by the specified icon if the memory is purgeable. (Deprecated. There is no replacement; this function was included to facilitate porting classic applications to Carbon, but it serves no useful purpose in Mac OS X.)

`FlushIconRefsByVolume` (page 1243) Deprecated in Mac OS X v10.3
On a given volume, reclaims memory used by purgeable icons. (Deprecated. There is no replacement; this function was included to facilitate porting classic applications to Carbon, but it serves no useful purpose in Mac OS X.)

Getting and Setting the Label for an Icon Suite

`GetSuiteLabel` (page 1259) Deprecated in Mac OS X v10.5
Gets the default label setting associated with an icon suite. (Deprecated. Use Icon Services instead.)

`SetSuiteLabel` (page 1300) Deprecated in Mac OS X v10.5
Specifies the default label associated with an icon suite. (Deprecated. Use Icon Services instead.)

Getting Label Information

`GetLabel` (page 1258) Deprecated in Mac OS X v10.5
Gets the color and string used for a given label in the Label menu of the Finder and in the Labels control panel. (Deprecated. Use Icon Services instead.)

Getting Icons From an Icon Suite

`GetIconFromSuite` (page 1249) Deprecated in Mac OS X v10.5
Gets an icon from an icon suite. (Deprecated. Use Icon Services instead.)

Getting Icons From Resources That Don't Belong to an Icon Family

`GetCIcon` (page 1244) Deprecated in Mac OS X v10.5
Gets a handle to a color icon of resource type `'cicn'`. (Deprecated. Use Icon Services instead.)

`GetIcon` (page 1246) Deprecated in Mac OS X v10.5
Gets a handle to an icon resource of type `'ICON'`. (Deprecated. Use Icon Services instead.)

IconRef Reference Counting

`AcquireIconRef` (page 1238)
Increments the reference count for an `IconRef`.

`GetIconRefOwners` (page 1255)
Provides the current reference count for an `IconRef`.

`ReleaseIconRef` (page 1296)
Decrements the reference count for an `IconRef`.

Modifying IconRef Data

`OverrideIconRef` (page 1274)
Replaces the bitmaps of one `IconRef` with those of another `IconRef`.

`RemoveIconRefOverride` (page 1297)
Restores the original bitmaps of an overridden `IconRef`.

`UpdateIconRef` (page 1301)
Forces an update of `IconRef` data.

`OverrideIconRefFromResource` (page 1275) Deprecated in Mac OS X v10.5
Replaces the bitmaps in an `IconRef` with bitmaps from a specified resource file. (Deprecated. Use `OverrideIconRef` (page 1274) instead.)

Obtaining Icon Data

`IsDataAvailableInIconRef` (page 1268)
Indicates whether an `IconRef` has the specified data.

`IsIconRefComposite` (page 1269)
Reports whether a specified `IconRef` has been composited.

`IsIconRefMaskEmpty` (page 1270)
Reports whether a specified mask is empty.

`IsValidIconRef` (page 1270)
Reports whether a specified `IconRef` is valid.

`GetIconSizesFromIconRef` (page 1256) Deprecated in Mac OS X v10.3
Provides an `IconSelectorValue` indicating the sizes and depths of icon data available for an `IconRef`. (Deprecated. Use `IsDataAvailableInIconRef` (page 1268) instead.)

Obtaining IconRef Values

`GetIconRef` (page 1249)
Provides an `IconRef` object for an icon in the desktop database or for a registered icon.

`GetIconRefFromFolder` (page 1252)
Provides an `IconRef` object for a folder with no custom icon.

`GetIconRefFromFileInfo` (page 1251)
Provides an `IconRef` object for a file with minimal file I/O.

`GetIconRefFromTypeInfo` (page 1254)
Provides an `IconRef` object with the specified type information.

`GetIconRefFromIconFamilyPtr` (page 1253)
Provides an `IconRef` object from a specified icon family.

`GetIconRefFromComponent` (page 1250)
Provides an `IconRef` object based on a specified component.

`GetIconRefFromFile` (page 1251) Deprecated in Mac OS X v10.5
Provides an `IconRef` object for a file, folder or volume. (Deprecated. Use `GetIconRefFromFileInfo` (page 1251) instead.)

Performing Operations on Icons in an Icon Suite

`ForEachIconDo` (page 1244) Deprecated in Mac OS X v10.5
Performs an action on one or more icons in an icon suite. (Deprecated. Use Icon Services instead.)

Reading, Copying, and Converting Icon Data

`GetIconFamilyData` (page 1248)
Obtains a copy of the raw icon data for an individual element in an icon family.

`IconRefToIconFamily` (page 1265)
Provides icon family data for a given `IconRef`.

`ReadIconFromFSRef` (page 1289)
Reads an icon ('`icns`') file into memory.

`SetIconFamilyData` (page 1299)
Provides new raw icon data for an individual element of an icon family.

`IconFamilyToIconSuite` (page 1259) Deprecated in Mac OS X v10.5
Provides icon suite data for a given icon family. (Deprecated. Use Icon Services instead.)

`IconSuiteToIconFamily` (page 1266) Deprecated in Mac OS X v10.5
Provides `IconFamily` data for a specified `IconSuite`. (Deprecated. Use Icon Services instead.)

`ReadIconFile` (page 1288) Deprecated in Mac OS X v10.5
Copies data from a given file into an icon family. (Deprecated. Use `ReadIconFromFSRef` (page 1289) instead.)

`WriteIconFile` (page 1302) Deprecated in Mac OS X v10.5
Copies data from a given icon family into a file. (Deprecated. Use the File Manager instead.)

Registering and Unregistering IconRef Values

`RegisterIconRefFromFSRef` (page 1293)
Registers an `IconRef` from a `.icns` file and associates it with a creator and type pair.

`RegisterIconRefFromIconFamily` (page 1294)
Adds an `iconFamily`-derived `IconRef` to the Icon Services registry.

`UnregisterIconRef` (page 1301)
Removes the specified icon data from the icon registry.

`RegisterIconRefFromIconFile` (page 1295) Deprecated in Mac OS X v10.5
Adds a file-derived `IconRef` to the Icon Services registry. (Deprecated. Use `RegisterIconRefFromFSRef` (page 1293) instead.)

`RegisterIconRefFromResource` (page 1295) Deprecated in Mac OS X v10.5
Adds a resource-derived `IconRef` to the Icon Services registry. (Deprecated. Use `RegisterIconRefFromFSRef` (page 1293) instead.)

Using IconRef Data

`CompositeIconRef` (page 1239)
Superimposes one `IconRef` onto another.

`GetIconRefVariant` (page 1255)
Specifies a transformation for a given `IconRef`.

`IconRefContainsCGPoint` (page 1262)
Returns a Boolean value indicating whether an icon contains a specified point.

`IconRefIntersectsCGRect` (page 1263)
Returns a Boolean value indicating whether an icon intersects a specified rectangle.

`IconRefToHIShape` (page 1264)
Converts an icon into an HIShape object.

`PlotIconRefInContext` (page 1281)
Plots an `IconRef` using Quartz.

`IconRefToRgn` (page 1265) Deprecated in Mac OS X v10.5
Converts an Icon Services icon into a QuickDraw region. (Deprecated. Use `IconRefToHIShape` (page 1264) instead.)

`PlotIconRef` (page 1281) Deprecated in Mac OS X v10.5
Draws an icon using appropriate size and depth data from an `IconRef`. (Deprecated. Use `PlotIconRefInContext` (page 1281) instead.)

`PtInIconRef` (page 1287) Deprecated in Mac OS X v10.5
Tests whether a specified point falls within an icon's mask. (Deprecated. Use `IconRefContainsCGPoint` (page 1262) instead.)

`RectInIconRef` (page 1292) Deprecated in Mac OS X v10.5
Tests whether a specified rectangle falls within an icon's mask. (Deprecated. Use `IconRefIntersectsCGRect` (page 1263) instead.)

Working With Icon Caches

`GetIconCacheData` (page 1246) Deprecated in Mac OS X v10.5
Gets the data associated with an icon cache. (Deprecated. Use Icon Services instead.)

`GetIconCacheProc` (page 1247) Deprecated in Mac OS X v10.5
Gets the icon getter function associated with an icon cache. (Deprecated. Use Icon Services instead.)

`LoadIconCache` (page 1270) Deprecated in Mac OS X v10.5
Loads into an icon cache a handle to the appropriate icon data for a specified destination rectangle and the current bit depth, for drawing later with a specified alignment and transform. (Deprecated. Use Icon Services instead.)

`MakeIconCache` (page 1272) Deprecated in Mac OS X v10.5
Gets a handle to an empty icon cache. (Deprecated. Use Icon Services instead.)

`SetIconCacheData` (page 1298) Deprecated in Mac OS X v10.5
Sets the data associated with an icon cache. (Deprecated. Use Icon Services instead.)

`SetIconCacheProc` (page 1299) Deprecated in Mac OS X v10.5
Sets the icon getter callback function associated with an icon cache. (Deprecated. Use Icon Services instead.)

Creating and Managing Universal Procedure Pointers

`NewIconActionUPP` (page 1272)
Creates a new universal procedure pointer (UPP) to an icon action callback function.

`NewIconGetterUPP` (page 1273)
Creates a new universal procedure pointer (UPP) to an icon getter callback function.

`DisposeIconActionUPP` (page 1241)
Disposes of the universal procedure pointer (UPP) to your icon action callback function.

`DisposeIconGetterUPP` (page 1241)
Disposes of the universal procedure pointer (UPP) to your icon getter callback function.

`InvokeIconActionUPP` (page 1268)
Calls your icon action callback function.

`InvokeIconGetterUPP` (page 1268)
Calls your icon getter callback function.

Functions

AcquireIconRef

Increments the reference count for an `IconRef`.

```
OSErr AcquireIconRef (
   IconRef theIconRef
);
```

Parameters

`theIconRef`
An `IconRef` whose reference count you wish to increment.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

AddIconToSuite

Adds an icon to an icon suite. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr AddIconToSuite (
   Handle theIconData,
   IconSuiteRef theSuite,
   ResType theType
);
```

Parameters

theIconData

A handle to the data for the new icon to be added to the icon suite. You can obtain a handle to icon data using various functions, such as `GetIcon` (page 1246) or `GetResource`.

The handle to the icon data is added at the location reserved for icon data of the type specified by `theType`. If the icon suite already includes a handle to icon data for that type, this function replaces the handle to the old data without disposing of it. In this case you may want to call the `GetIconFromSuite` (page 1249) function first to obtain the old handle so that you can dispose of it.

The handles that you add to the suite do not have to be associated with a resource fork. For example, your application might get icon data from the desktop database rather than reading it from a resource, or your application might read icon data from a resource and then detach it.

theSuite

A handle to the icon suite to which to add the icon.

theType

The resource type of the new icon. The resource type should be that of an icon family member.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

This function is most often used to read icons into an empty icon suite created with the `NewIconSuite` (page 1273) function.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

CompositeIconRef

Superimposes one `IconRef` onto another.

```
OSErr CompositeIconRef (
   IconRef backgroundIconRef,
   IconRef foregroundIconRef,
   IconRef *compositeIconRef
);
```

Parameters

backgroundIconRef

A value to use as the background for the composite `IconRef`.

foregroundIconRef

A value to use as the foregound for the composite `IconRef`.

compositeIconRef

On completion, this points to an `IconRef` that is a composite of the specified background and foreground `IconRefs`.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
This function provides an alternative to badging when you need to indicate a change of state.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

DisposeCIcon

Releases the memory occupied by a color icon structure. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
void DisposeCIcon (
   CIconHandle theIcon
);
```

Parameters

theIcon

A handle to the color icon structure to dispose of, previously obtained from the `GetCIcon` (page 1244) function.

Discussion
To dispose of a handle obtained from `GetIcon` or `GetResource`, use the `ReleaseResource` function to release the memory occupied by the icon resource data.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Icons.h`

DisposeIconActionUPP

Disposes of the universal procedure pointer (UPP) to your icon action callback function.

```
void DisposeIconActionUPP (
   IconActionUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Discussion
See the `IconActionProcPtr` (page 1302) callback for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

DisposeIconGetterUPP

Disposes of the universal procedure pointer (UPP) to your icon getter callback function.

```
void DisposeIconGetterUPP (
   IconGetterUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Discussion
See the `IconGetterProcPtr` (page 1303) callback for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

DisposeIconSuite

Releases the memory occupied by an icon suite. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr DisposeIconSuite (
   IconSuiteRef theIconSuite,
   Boolean disposeData
);
```

Parameters

theIconSuite

A handle to the icon suite to be disposed of.

disposeData

A Boolean value indicating whether or not to dispose of handles in the icon suite that are not associated with a resource fork.

Set this value to `TRUE` to automatically release icon data that is associated with the specified icon suite but not explicitly associated with a resource fork. If you set this value to `FALSE`, the function does not dispose of any icon data that is associated with the specified icon suite.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

This function does not release the memory of any icons explicitly associated with an open resource fork, that is, any handles to icon resource data that your application added to the suite using the functions `GetIconSuite` (page 1257) or `AddIconToSuite` (page 1238). For handles to icon data that your application added to the icon suite using `AddIconToSuite` (for example, if your application read in an icon resource, detached it, then added the handle to the suite), you can request that `AddIconToSuite` release the memory associated with the handles.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

FlushIconRefs

Reclaims memory used by the specified icon if the memory is purgeable. (Deprecated in Mac OS X v10.3. There is no replacement; this function was included to facilitate porting classic applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
OSErr FlushIconRefs (
   OSType creator,
   OSType iconType
);
```

Parameters

creator

The creator code of the file whose icon data is to be flushed.

iconType
The type code of the file whose icon data is to be flushed.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function does nothing in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`IconsCore.h`

FlushIconRefsByVolume

On a given volume, reclaims memory used by purgeable icons. (Deprecated in Mac OS X v10.3. There is no replacement; this function was included to facilitate porting classic applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
OSErr FlushIconRefsByVolume (
   SInt16 vRefNum
);
```

Parameters

vRefNum
The volume whose icon cache is to be flushed.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Calling this function locks the bitmap data of all `IconRefs` with non-zero reference counts (that is, all `IconRefs` that are in use) on the volume. The Finder normally maintains a number of `IconRefs` with non-zero reference counts, so you should use the function `FlushIconRefs` (page 1242) instead of the `FlushIconRefsByVolume` function whenever feasible.

Special Considerations

This function does nothing in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`IconsCore.h`

ForEachIconDo

Performs an action on one or more icons in an icon suite. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr ForEachIconDo (
   IconSuiteRef theSuite,
   IconSelectorValue selector,
   IconActionUPP action,
   void *yourDataPtr
);
```

Parameters

theSuite

A handle to an icon suite.

selector

Indicates which icons in the suite to perform the operation on. See "Icon Selector Constants" (page 1310) for a description of the values you can use in this parameter.

action

A universal procedure pointer to your icon action callback function. The `ForEachIconDo` function uses this icon action function to perform an action on the specified icons in the icon suite.

`ForEachIconDo` calls your icon action function once for each type of icon specified in the `selector` parameter. `ForEachIconDo` passes to your icon action function a handle to the icon to perform the action on. Your icon action function should perform any action as indicated by the `yourDataPtr` parameter and return a result code.

See the `IconActionProcPtr` (page 1302) callback for more information about icon action functions.

yourDataPtr

A pointer to data or other information required by your icon action function that is passed to your icon action function. Typically, you use this parameter to specify which action your icon action function should perform.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324). The result code returned by your icon action function. If your icon action function returns a nonzero function result, `ForEachIconDo` immediately returns to the application.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

GetCIcon

Gets a handle to a color icon of resource type `'cicn'`. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
CIconHandle GetCIcon (
   SInt16 iconID
);
```

Parameters

iconID

The resource ID for an icon of resource type `'cicn'`. In general, you should specify your icon resources as purgeable.

Return Value

A handle to the `CIcon` (page 1305) structure for the icon, or `NULL` if the function could not find the resource.

Discussion

The function searches the current resource chain for the resource. If it finds the resource, it reads the resource, creates a color icon structure for the icon, and initializes the fields of the structure according to the information contained in the `'cicn'` resource.

To draw an icon obtained from this function in a specified rectangle, you can use either the `PlotCIcon` (page 1275) function, or the `PlotCIconHandle` (page 1276) function. The latter function allows you to specify transforms and alignments.

When you are finished with a handle obtained from this function, use the `DisposeCIcon` (page 1240) function to release the memory occupied by the color icon structure.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

GetCustomIconsEnabled

Determines whether custom icons are enabled or disabled on a specified volume.

```
OSErr GetCustomIconsEnabled (
   SInt16 vRefNum,
   Boolean *customIconsEnabled
);
```

Parameters

vRefNum

The volume whose status you are querying.

customIconsEnabled

On return, `customIconsEnabled` points to the value `true` if custom icons are enabled on the volume specified or `false` if custom icons are disabled on the volume specified.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

GetIcon

Gets a handle to an icon resource of type `'ICON'`. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
Handle GetIcon (
   SInt16 iconID
);
```

Parameters

iconID

The resource ID for an icon of resource type `'ICON'`. The function searches the current resource chain for the resource. In general, you should specify your icon resources as purgeable.

Return Value
A handle to the icon with the specified ID or `NULL` if the function could not find the resource.

Discussion
To draw an icon obtained from this function in a specified rectangle, you can use either the `PlotIcon` (page 1277) function, or the `PlotIconHandle` (page 1278) function. The latter function allows you to specify transforms and alignments.

When you are finished using a handle obtained from this function, use the `ReleaseResource` function to release the memory occupied by the icon resource data.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

GetIconCacheData

Gets the data associated with an icon cache. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr GetIconCacheData (
   IconCacheRef theCache,
   void **theData
);
```

Parameters

theCache

A handle to the icon cache whose data is desired.

theData

On return, a pointer to a pointer to the data associated with the icon cache.

You associate data with an icon cache when you first create the cache using the `MakeIconCache` (page 1272) function. You can also set this data using the `SetIconCacheData` (page 1298) function.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
All the Icon Utilities functions that accept a handle to an icon suite also accept a handle to an icon cache.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

GetIconCacheProc

Gets the icon getter function associated with an icon cache. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr GetIconCacheProc (
   IconCacheRef theCache,
   IconGetterUPP *theProc
);
```

Parameters

theCache

A handle to the icon cache whose associated icon getter function is desired.

theProc

On return, a pointer to the universal procedure pointer to the icon getter callback function associated with the specified cache. See the `IconGetterProcPtr` (page 1303) callback for more information on icon getter functions.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
All the Icon Utilities functions that accept a handle to an icon suite also accept a handle to an icon cache. An icon cache is like an icon suite except that it also contains a pointer to an icon getter callback function and a pointer to data that can be used as a reference constant. An icon cache typically does not contain handles to the icon resources for all icon family members. Instead, if the icon cache does not contain an entry for a specific type of icon in an icon family, the Icon Utilities functions call your application's icon getter function to retrieve the data for that icon type.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Icons.h`

GetIconFamilyData

Obtains a copy of the raw icon data for an individual element in an icon family.

```
OSErr GetIconFamilyData (
   IconFamilyHandle iconFamily,
   OSType iconType,
   Handle h
);
```

Parameters

iconFamily

A handle to an `iconFamily` data structure to use as a source for icon data.

iconType

The format of the icon data you want to obtain.You may specify one of the icon types (as defined in `IconStorage.h` in the CoreServices/OSServices framework) or `'PICT'` in this parameter. For example, you can pass `kThumbnail32BitData` (`'it32'`), to obtain 65,536 bytes of raw bitmap data.

h

A handle to the icon data being returned. Icon Services resizes this handle as needed. If no data is available for the specified icon family, Icon Services sets the handle to 0.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

GetIconFromSuite

Gets an icon from an icon suite. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr GetIconFromSuite (
   Handle *theIconData,
   IconSuiteRef theSuite,
   ResType theType
);
```

Parameters

theIconData

On return, a pointer to a handle to the data for the requested icon. If an icon of the specified type does not exist in the given icon suite, this parameter is `NULL`.

If you intend to dispose of the handle, pass a `NULL` handle to the `AddIconToSuite` (page 1238) function to delete the corresponding entry in the suite.

You can use the handle returned by this function to manipulate the icon data, for example, to alter its color or add three-dimensional shading. However, you should not use the returned handle to draw the icon with other Icon Utilities functions.

To plot an icon from an icon suite, you should normally use the `PlotIconSuite` (page 1282) function. The `PlotIconHandle` (page 1278) function may not draw the icon correctly if you pass it the handle returned in this parameter.

theSuite

A handle to the icon suite from which to get the icon.

theType

The resource type of the desired icon.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

GetIconRef

Provides an `IconRef` object for an icon in the desktop database or for a registered icon.

```
OSErr GetIconRef (
   SInt16 vRefNum,
   OSType creator,
   OSType iconType,
   IconRef *theIconRef
);
```

Parameters

vRefNum

The volume where Icon Services should start to search for the desired icon. Pass the `kOnSystemDisk` constant if you are not sure which value to specify in this parameter.

creator

The creator code of the desired icon.

iconType

The type code of the desired icon.

theIconRef

On return, a pointer to an `IconRef` object. You are responsible for releasing the object by calling `ReleaseIconRef` (page 1296).

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Icon Services defines constants for commonly-used system icons. You can pass one of these constants in the `iconType` parameter if you specify `kSystemIconsCreator` in the `creator` parameter. See "Folder Icon Constants" (page 1315) for a list of these constants.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

GetIconRefFromComponent

Provides an `IconRef` object based on a specified component.

```
OSStatus GetIconRefFromComponent (
   Component inComponent,
   IconRef *outIconRef
);
```

Parameters

inComponent

The component whose icon data you want to obtain.

outIconRef

On return, a pointer to an `IconRef` object based on the `componentIconFamily` field of the specified component's `'thng'` resource. You are responsible for releasing the object by calling `ReleaseIconRef` (page 1296).

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
This function obtains an `IconRef` object based on the resource ID of an icon family. A component can provide an icon family in addition to the icon provided in the `componentIcon` field. Note that members of this icon family are not used by the Finder; you supply an icon family only so that other components or applications can display your component's icon in their user interfaces if needed.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`IconsCore.h`

GetIconRefFromFile

Provides an `IconRef` object for a file, folder or volume. (Deprecated in Mac OS X v10.5. Use `GetIconRefFromFileInfo` (page 1251) instead.)

```
OSErr GetIconRefFromFile (
   const FSSpec *theFile,
   IconRef *theIconRef,
   SInt16 *theLabel
);
```

Parameters

theFile
A pointer to the `FSSpec` structure specifying the file, folder or volume for the `IconRef`.

theIconRef
On return, a pointer to an `IconRef` object. You are responsible for releasing the object by calling `ReleaseIconRef` (page 1296).

theLabel
On return, a pointer to the file or folder's label.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Use this function if you have no information about the file object passed in the `theFile` parameter. If you have already called the File System Manager function `PBGetCatInfo`, you can use the function `GetIconRefFromFolder` (page 1252) if the object is a folder without custom icons or the function `GetIconRef` (page 1249) if the object is a file without custom icons.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`IconsCore.h`

GetIconRefFromFileInfo

Provides an `IconRef` object for a file with minimal file I/O.

```
OSStatus GetIconRefFromFileInfo (
    const FSRef *inRef,
    UniCharCount inFileNameLength,
    const UniChar *inFileName,
    FSCatalogInfoBitmap inWhichInfo,
    const FSCatalogInfo *inCatalogInfo,
    IconServicesUsageFlags inUsageFlags,
    IconRef *outIconRef,
    SInt16 *outLabel
);
```

Parameters

inRef

A pointer to an `FSRef` for the target file.

inFileNameLength

The length of the name of the target file.

inFileName

A pointer to the name of the target file.

inWhichInfo

The mask of the file information contained in the *inCatalogInfo* parameter.

inCatalogInfo

A pointer to the catalog information.

inUsageFlags

The usage flags for this call; use `kIconServicesNormalUsageFlag`.

outIconRef

On return, a pointer to an `IconRef` object. You are responsible for releasing the object by calling `ReleaseIconRef` (page 1296).

outLabel

On return, a pointer to the output label for the icon/file.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

To minimize file operations, `FSGetCatalogInfo` should be called prior to calling this function. The information in the `FSCatalogInfo` structure should correspond to that specified by `kIconServicesCatalogInfoMask`. The name should be fetched and passed in. If either the name or the correct catalog information is not passed in, this function will do file operations for this information instead.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`IconsCore.h`

GetIconRefFromFolder

Provides an `IconRef` object for a folder with no custom icon.

```
OSErr GetIconRefFromFolder (
   SInt16 vRefNum,
   SInt32 parentFolderID,
   SInt32 folderID,
   SInt8 attributes,
   SInt8 accessPrivileges,
   IconRef *theIconRef
);
```

Parameters

vRefNum

The volume where the folder is located.

parentFolderID

The ID of the desired folder's parent folder.

folderID

The ID of the desired folder.

attributes

The attributes of the desired folder. You can obtain this data from the `CInfoPBRec.dirInfo.ioFlAttrib` field of the folder's catalog information record.

accessPrivileges

The access privileges of the specified folder. You can obtain this data from the `CInfoPBRec.dirInfo.ioACUser` field of the folder's catalog information record.

theIconRef

On return, a pointer to an `IconRef` object. You are responsible for releasing the object by calling `ReleaseIconRef` (page 1296).

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

If you do not have the catalog information for a folder, use the function `GetIconRefFromFileInfo` (page 1251).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`IconsCore.h`

GetIconRefFromIconFamilyPtr

Provides an `IconRef` object from a specified icon family.

```
OSStatus GetIconRefFromIconFamilyPtr (
    const IconFamilyResource *inIconFamilyPtr,
    Size inSize,
    IconRef *outIconRef
);
```

Parameters

inIconFamilyPtr

A pointer to an icon family. See `IconStorage.h` for more information.

inSize
The size of the resource buffer containing the icon family.

outIconRef
On return, a pointer to an `IconRef` object that matches the specified inputs. You are responsible for releasing the object by calling `ReleaseIconRef` (page 1296).

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Typically, you do not need to use this function.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`IconsCore.h`

GetIconRefFromTypeInfo

Provides an `IconRef` object with the specified type information.

```
OSErr GetIconRefFromTypeInfo (
    OSType inCreator,
    OSType inType,
    CFStringRef inExtension,
    CFStringRef inMIMEType,
    IconServicesUsageFlags inUsageFlags,
    IconRef *outIconRef
);
```

Parameters

inCreator
The creator code of the desired `IconRef`. You may pass 0 if the creator code is unknown.

inType
The type code of the desired `IconRef`. You may pass 0 if the type code is unknown.

inExtension
The file name extension of the desired `IconRef`. You may pass `NULL` if the extension is unknown.

inMIMEType
The MIME type of the desired `IconRef`. You may pass `NULL` if the MIME type is unknown.

inUsageFlags
The usage flags; use `kIconServicesNormalUsageFlag`.

outIconRef
On return, a pointer to an `IconRef` object that most closely matches the specified inputs. You are responsible for releasing the object by calling `ReleaseIconRef` (page 1296).

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
This function serves as a more versatile version of `GetIconRef` (page 1249). If you specify creator and type codes and do not specify the extension and MIME type, calling this function is equivalent to calling `GetIconRef (kOnSystemDisk, inCreator, inType)`. If none of the input parameters is specified or if no match is found, this function returns the generic document icon.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`IconsCore.h`

GetIconRefOwners

Provides the current reference count for an `IconRef`.

```
OSErr GetIconRefOwners (
   IconRef theIconRef,
   UInt16 *owners
);
```

Parameters

`theIconRef`
An `IconRef` whose reference count you wish to obtain.

`owners`
On return, a pointer to the value which represents the current reference count.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
When an `IconRef`'s reference count reaches 0, all memory allocated for the `IconRef` is marked as disposable. Any subsequent attempt to use the `IconRef` returns a result code of `-2580` (`invalidIconRefErr`).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

GetIconRefVariant

Specifies a transformation for a given `IconRef`.

```
IconRef GetIconRefVariant (
   IconRef inIconRef,
   OSType inVariant,
   IconTransformType *outTransform
);
```

Parameters

`inIconRef`
A value to be tested.

inVariant

A four-character value. You specify a variant by passing one of the following constants:

`kTileIconVariant` specifies a tiled icon.

`kRolloverIconVariant` specifies a rollover icon.

`kDropIconVariant` specifies a drop target icon.

`kOpenIconVariant` specifies an open icon.

`kOpenDropIconVariant` specifies a open drop target icon.

outTransform

On completion, this points to a transformation type that you pass to the function `PlotIconRef` (page 1281) for purposes of hit-testing.

Return Value

An `IconRef` value that you pass to the function `PlotIconRef` (page 1281) for purposes of hit-testing.

Discussion

Icon variants give you a simple way to indicate a temporary change of state by changing an icon's appearance. For example, if you specify the `kDropIconVariant` value when the user drags over a valid drop target, the `GetIconVariant` function provides the appropriate data for you to plot the variant with the function `PlotIconRef` (page 1281).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Icons.h`

GetIconSizesFromIconRef

Provides an `IconSelectorValue` indicating the sizes and depths of icon data available for an `IconRef`. (Deprecated in Mac OS X v10.3. Use `IsDataAvailableInIconRef` (page 1268) instead.)

```
OSErr GetIconSizesFromIconRef (
   IconSelectorValue iconSelectorInput,
   IconSelectorValue *iconSelectorOutputPtr,
   IconServicesUsageFlags iconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

iconSelectorInput

The icon sizes and depths you are requesting from the `IconRef`. For a description of the possible values, see "Icon Selector Constants" (page 1310).

iconSelectorOutputPtr

On return, this points to a value describing the icon sizes and depths available for the specified `IconRef`. For a description of the possible values, see "Icon Selector Constants" (page 1310).

iconServicesUsageFlags

Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef

The icon family to query.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Note that this function may be very time-consuming, as Icon Services may have to search disks or even the network to obtain the requested data.

Special Considerations

Because this function is so time-consuming, it is more efficient to simply query the icon for particular data using the function `IsDataAvailableInIconRef` (page 1268).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`Icons.h`

GetIconSuite

Creates an icon suite in memory that contains handles to a specified icon family's resources. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr GetIconSuite (
   IconSuiteRef *theIconSuite,
   SInt16 theResID,
   IconSelectorValue selector
);
```

Parameters

theIconSuite
On return, a pointer to a handle to an icon suite for the requested icon family, for which this function allocates the memory. To release the memory occupied by an icon suite, you must use the `DisposeIconSuite` function.

theResID
The resource ID of the icons in the icon family to be read into memory. In general, you should specify your icon resources as purgeable.

selector
Indicates which icons from the icon family to include in the icon suite. See "Icon Selector Constants" (page 1310) for a description of the values you can use in this parameter.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
When you create an icon suite from icon family resources, the associated resource file should remain open while you use Icon Utilities functions. If you call the `SetResLoad` function with the `load` parameter set to `FALSE` before you call this function, the suite is filled with unloaded resource handles.

When you create an icon suite using this function, it sets the default label for the suite to none. To set a new default label for an icon suite, use the `SetSuiteLabel` (page 1300) function. To perform operations on one or more icons in an icon suite, use the `ForEachIconDo` (page 1244) function. To draw the icon described by the icon suite using the icon family member that is most suitable for the current bit depth of the display device, use the `PlotIconSuite` (page 1282) function.

As an alternative to this function, you can also create an empty icon suite using the `NewIconSuite` (page 1273) function and then add icons to it one at a time using the `AddIconToSuite` (page 1238) function.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

GetLabel

Gets the color and string used for a given label in the Label menu of the Finder and in the Labels control panel. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr GetLabel (
   SInt16 labelNumber,
   RGBColor *labelColor,
   Str255 labelString
);
```

Parameters

labelNumber
An integer from 1 to 7 indicating which label's information is requested.

labelColor
On return, a pointer to the color of the specified label.

labelString
On return, the string associated with the specified label.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

GetSuiteLabel

Gets the default label setting associated with an icon suite. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
SInt16 GetSuiteLabel (
   IconSuiteRef theSuite
);
```

Parameters

theSuite
A handle to an icon suite.

Return Value
The default label setting associated with the specified icon suite. The default label setting is an integer from 1 to 7 that specifies which of the label colors shown in the Finder's Label menu is applied to icons of that suite when your application displays them. The function returns 0 if the suite doesn't have a label. You can override the default label setting for a suite by specifying a label in the `transform` parameter of the `PlotIconSuite` (page 1282) function. To get information about the color and string for a specific label, you can use the `GetLabel` (page 1258) function.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

IconFamilyToIconSuite

Provides icon suite data for a given icon family. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr IconFamilyToIconSuite (
   IconFamilyHandle iconFamily,
   IconSelectorValue whichIcons,
   IconSuiteRef *iconSuite
);
```

Parameters

iconFamily
A handle to an `iconFamily` data structure to use as a source for icon data. For more information on the `IconFamily` data structure, see 'icns'.

whichIcons

The depths and sizes of icons to extract from the IconFamily data structure. For a description of the possible values, see "Icon Selector Constants" (page 1310).

iconSuite

On return, a pointer to the structure which contains icon data as specified in the `iconFamily` and `whichIcons` parameters. Icon Services returns `NULL` if no appropriate icon data is found.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

IconIDToRgn

Converts the icon mask in an icon family to a region. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr IconIDToRgn (
   RgnHandle theRgn,
   const Rect *iconRect,
   IconAlignmentType align,
   SInt16 iconID
);
```

Parameters

theRgn

On return, a handle to the requested region. You must allocate memory for the region handle before calling this function.

The returned region corresponds to the icon's mask (the mask defined by either an `'ICN#'` or `'ics#'` resource in an icon family, according to the rectangle and alignment specified in the `iconRect` and `align` parameters).

Once you have a region that describes the icon mask for a given icon, you can use it to perform accurate hit-testing and outline dragging of the icon in your application.

iconRect

A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port. The function uses this rectangle as the bounding box of the region. The function determines, from the size of the rectangle specified here, which icon mask to use from the specified icon family.

align

Specifies how the function should align the mask within the rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

iconID

The resource ID of the icon for which to create a region. In general, you should specify your icon resources as purgeable.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

IconMethodToRgn

Converts, to a region, the mask for an icon that `IconMethodToRgn` obtains with the aid of your icon getter callback function. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr IconMethodToRgn (
   RgnHandle theRgn,
   const Rect *iconRect,
   IconAlignmentType align,
   IconGetterUPP theMethod,
   void *yourDataPtr
);
```

Parameters

theRgn

On return, a handle to the requested region. You must allocate memory for the region handle before calling this function. Once you have a region that describes the icon mask for a given icon, you can use it to perform accurate hit-testing and outline dragging of the icon in your application.

iconRect

A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port. The function obtains the data for the icon mask from your icon getter function and then converts the icon mask to a region. The function uses the rectangle specified in this parameter as the bounding box of the region.

align

Specifies how the function should align the mask within the rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

theMethod

A universal procedure pointer to your icon getter callback function. `IconMethodToRgn` passes to your icon getter function the type of the icon to get and the value specified in the `yourDataPtr` parameter. The `IconMethodToRgn` function examines the size of the rectangle and requests the appropriate icon from your icon getter function—an icon of icon type `'ICN#'` or `'ics#'`. Your icon getter function should return a handle to the data of the requested icon type. The `IconMethodToRgn` function extracts the mask from the icon data that your icon getter function returns. If your icon getter function returns data that does not correspond to an icon of type `'ICN#'` or type `'ics#'`, `IconMethodToRgn` attempts to generate a mask from the returned data.

Your icon getter function can get the data for the icon and its mask using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons (and pass a pointer to it in the `yourDataPtr` parameter) or use its icon getter function to get an icon from the desktop database.

See the `IconGetterProcPtr` (page 1303) callback for more information on creating an icon getter function.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Icons.h`

IconRefContainsCGPoint

Returns a Boolean value indicating whether an icon contains a specified point.

```
Boolean IconRefContainsCGPoint (
   const CGPoint *testPt,
   const CGRect *iconRect,
   IconAlignmentType align,
   IconServicesUsageFlags iconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

testPt

A pointer to the point to be tested. The point should be specified in the coordinate system of the rectangle specified in the `iconRect` parameter.

iconRect

A pointer to the rectangle in which the icon appears. The rectangle you specify should be the same rectangle that you last used to draw the icon.

align
Specifies how the icon is aligned within the rectangle specified in the `iconRect` parameter. The alignment you specify should be the same alignment that you last used to draw the icon. See "`Icon Alignment Constants`" (page 1307) for a description of the values you can use in this parameter.

iconServicesUsageFlags
Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef
The icon to test.

Return Value
`TRUE` if the point is in the icon and `FALSE` if it is not.

Discussion
This function uses the size of the rectangle you specify to determine the optimal icon mask to represent the icon. The function uses the alignment information you specify to adjust the position of the mask inside its bounding rectangle, and then determines whether the specified point is within the mask.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Icons.h`

IconRefIntersectsCGRect

Returns a Boolean value indicating whether an icon intersects a specified rectangle.

```
Boolean IconRefIntersectsCGRect (
   const CGRect *testRect,
   const CGRect *iconRect,
   IconAlignmentType align,
   IconServicesUsageFlags iconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

testRect
A pointer to the rectangle to be tested. The rectangle should be specified in the coordinate system of the rectangle specified in the `iconRect` parameter.

iconRect
A pointer to the rectangle in which the icon appears. The rectangle you specify should be the same rectangle that you last used to draw the icon.

align
Specifies how the icon is aligned within the rectangle specified by the `iconRect` parameter. The alignment you specify should be the same alignment that you last used to draw the icon. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

iconServicesUsageFlags
Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef
The icon to test.

Return Value
`TRUE` if the point is in the icon and `FALSE` if it is not.

Discussion
This function uses the size of the rectangle you specify in the `iconRect` parameter to determine the optimal icon mask to represent the icon. The function uses the alignment information you specify to adjust the position of the mask inside its bounding rectangle, and then determines whether the rectangle you specify in the `testRect` parameter intersects the mask.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Icons.h`

IconRefToHIShape

Converts an icon into an HIShape object.

```
HIShapeRef IconRefToHIShape (
   const CGRect *iconRect,
   IconAlignmentType align,
   IconServicesUsageFlags iconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

iconRect
A pointer to the rectangle defining the area that Icon Services uses as the bounding box of the shape.

align
A value which determines how Icon Services aligns the shape within the rectangle. For a description of possible values, see "Icon Alignment Constants" (page 1307).

iconServicesUsageFlags
Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef
The icon to be converted.

Return Value
An HIShape object, or `NULL` if the icon could not be converted.

Discussion
This function uses the size of the rectangle you specify to determine the optimal icon mask to represent the icon. The function uses the alignment information you specify to adjust the position of the mask inside its bounding rectangle, and then returns the shape of the mask.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Icons.h`

IconRefToIconFamily

Provides icon family data for a given `IconRef`.

```
OSErr IconRefToIconFamily (
   IconRef theIconRef,
   IconSelectorValue whichIcons,
   IconFamilyHandle *iconFamily
);
```

Parameters

theIconRef

An `IconRef` to use as a source for icon data.

whichIcons

The depths and sizes of icons in the `iconFamily` data structure. For a description of the possible values, see "Icon Selector Constants" (page 1310).

iconFamily

On return, a pointer to a handle to the data structure which contains icon data as specified in the `IconRef` and `whichIcons` parameters. Icon Services returns `NULL` if no appropriate icon data is found. For more information on the `IconFamily` data structure, see 'icns'.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Icons.h`

IconRefToRgn

Converts an Icon Services icon into a QuickDraw region. (Deprecated in Mac OS X v10.5. Use `IconRefToHIShape` (page 1264) instead.)

```
OSErr IconRefToRgn (
   RgnHandle theRgn,
   const Rect *iconRect,
   IconAlignmentType align,
   IconServicesUsageFlags iconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

theRgn

A handle to the requested region. You must call the QuickDraw function `NewRegion` to allocate memory for the region handle before calling the `IconRefToRgn` function.

iconRect

A pointer to the rectangle defining the area that Icon Services uses as the bounding box of the region.

align

The value which determines how Icon Services aligns the region within the rectangle. For a description of possible return values, see "Icon Alignment Constants" (page 1307).

iconServicesUsageFlags
Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef
The `IconRef` for the icon family to use for drawing the requested region.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Icon Services uses the rectangle and alignment values to automatically select the icon used to generate the region data.

This function is similar to the Icon Utilities function `IconSuiteToRegion`.

Icon Services uses the icon's black-and-white mask to determine the region data, even if you provide a deep mask.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

IconSuiteToIconFamily

Provides `IconFamily` data for a specified `IconSuite`. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr IconSuiteToIconFamily (
   IconSuiteRef iconSuite,
   IconSelectorValue whichIcons,
   IconFamilyHandle *iconFamily
);
```

Parameters
iconSuite
The `IconSuiteRef` to use as a source for icon data.

whichIcons
The depths and sizes of icons to extract from the `iconFamily` data structure. For a description of the possible values, see "Icon Selector Constants" (page 1310).

iconFamily
On return, a pointer to a handle to the structure which contains icon data as specified in the `iconSuite` and `whichIcons` parameters. Icon Services returns `NULL` if no appropriate icon data is found. For more information on the `IconFamily` data structure, see 'icns'.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Icons.h`

IconSuiteToRgn

Converts the icon mask in an icon suite to a region. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr IconSuiteToRgn (
   RgnHandle theRgn,
   const Rect *iconRect,
   IconAlignmentType align,
   IconSuiteRef theIconSuite
);
```

Parameters

theRgn

On return, a handle to the requested region. You must allocate memory for the region handle before calling this function.

The returned region corresponds to the icon's mask (the mask defined by either an `'ICN#'` or `'ics#'` entry in an icon suite, according to the rectangle and alignment specified in the `iconRect` and `align` parameters).

Once you have a region that describes the icon mask for a given icon, you can use it to perform accurate hit-testing and outline dragging of the icon in your application.

iconRect

A pointer to the rectangle in which the icon is to be drawn, specified in local coordinates of the current graphics port. The function uses this rectangle as the bounding box of the region. The function determines, from the size of the rectangle specified here, which icon mask to use from the icon suite.

align

Specifies how the function should align the region within the rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

theIconSuite

A handle to an icon suite.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

InvokeIconActionUPP

Calls your icon action callback function.

```
OSErr InvokeIconActionUPP (
   ResType theType,
   Handle *theIcon,
   void *yourDataPtr,
   IconActionUPP userUPP
);
```

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
You should not need to use the function `InvokeIconActionUPP`, as the system calls your icon action callback function for you.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

InvokeIconGetterUPP

Calls your icon getter callback function.

```
Handle InvokeIconGetterUPP (
   ResType theType,
   void *yourDataPtr,
   IconGetterUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeIconGetterUPP`, as the system calls your icon getter callback function for you.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

IsDataAvailableInIconRef

Indicates whether an `IconRef` has the specified data.

```
Boolean IsDataAvailableInIconRef (
   OSType inIconKind,
   IconRef inIconRef
);
```

Parameters

inIconKind

The icon data kind. See `IconStorage.h` for more information.

inIconRef

The icon reference whose data you want to check.

Return Value

`True` if the icon reference contains the indicated data, `False` otherwise.

Discussion

This function can be used to determine the optimal icon size if you plan to cache a bitmap image of the icon.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`IconsCore.h`

IsIconRefComposite

Reports whether a specified `IconRef` has been composited.

```
OSErr IsIconRefComposite (
   IconRef compositeIconRef,
   IconRef *backgroundIconRef,
   IconRef *foregroundIconRef
);
```

Parameters

compositeIconRef

An `IconRef` that you wish to test to determine whether it has been composited.

backgroundIconRef

On return, this points to the `IconRef` value that forms the background of the `IconRef` specified in the `compositeIconRef` parameter. If the `IconRef` specified in the `compositeIconRef` parameter is not a composite, the return value is 0.

foregroundIconRef

On return, this points to the `IconRef` value that forms the foreground of the `IconRef` specified in the `compositeIconRef` parameter. If the `IconRef` specified in the `compositeIconRef` parameter is not a composite, the return value is 0.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

The function `CompositeIconRef` (page 1239) allows the creation of a composite `IconRef` from a given background `IconRef` and a given foreground `IconRef`. The `IsIconRefComposite` function checks a specified `IconRef` to determine whether it is a composite and, if so, provides the background and foreground `IconRef` values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

IsIconRefMaskEmpty

Reports whether a specified mask is empty.

```
Boolean IsIconRefMaskEmpty (
   IconRef iconRef
);
```

Parameters

`iconRef`
An `IconRef` whose mask you wish to test.

Return Value
`true` if the mask associated with the given `IconRef` is empty, `false` otherwise.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

IsValidIconRef

Reports whether a specified `IconRef` is valid.

```
Boolean IsValidIconRef (
   IconRef theIconRef
);
```

Parameters

`theIconRef`
An `IconRef`.

Return Value
`true` if the specified `IconRef` is valid, `false` otherwise.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

LoadIconCache

Loads into an icon cache a handle to the appropriate icon data for a specified destination rectangle and the current bit depth, for drawing later with a specified alignment and transform. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr LoadIconCache (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   IconCacheRef theIconCache
);
```

Parameters

theRect

A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port. The function uses the rectangle specified in this parameter and the bit depth of the display device to determine which icon type to load into the cache.

align

Specifies how to align the icon within the rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

transform

Specifies how to modify the appearance of the icon. See "Icon Transformation Constants" (page 1309) for a description of the values you can use in this parameter.

theIconCache

A reference to the icon cache into which to load the icon data.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

This function can be useful, for example, if you suspect that the icon may be drawn at a time not convenient for loading resource data (for instance, when the resource fork isn't in the current resource chain). The function uses the same criteria as the `PlotIconSuite` (page 1282) function to select the icon to load.

This function uses the icon getter callback function associated with the icon cache to get the appropriate icon. The icon getter function returns a handle to the requested icon data, and `LoadIconCache` adds the returned handle to the entry for that icon in the icon cache.

After calling this function, you can pass the same parameters to `PlotIconSuite` to plot the icon data. Note that if you specify an alignment when you call `LoadIconCache`, then call `PlotIconSuite` and specify no alignment, `PlotIconSuite` draws the icon using the alignment that you originally specified to `LoadIconCache`.

All the Icon Utilities functions that accept a handle to an icon suite also accept a handle to an icon cache.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

MakeIconCache

Gets a handle to an empty icon cache. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr MakeIconCache (
   IconCacheRef *theCache,
   IconGetterUPP makeIcon,
   void *yourDataPtr
);
```

Parameters

theCache

On return, a pointer to a handle to the new, empty icon cache. The function allocates the necessary memory. You can add icon data to the new cache using the `LoadIconCache` (page 1270) function.

makeIcon

A universal procedure pointer to the icon getter callback function to associate with the icon cache. See the `IconGetterProcPtr` (page 1303) callback for more information on icon getter callback functions.

yourDataPtr

A pointer to the data to associate with the icon cache.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

All the Icon Utilities functions that accept a handle to an icon suite also accept a handle to an icon cache. An icon cache is like an icon suite except that it also contains a pointer to an icon getter callback function and a pointer to data that can be used as a reference constant. An icon cache typically does not contain handles to the icon resources for all icon family members. Instead, if the icon cache does not contain an entry for a specific type of icon in an icon family, the Icon Utilities functions call your application's icon getter function to retrieve the data for that icon type.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

NewIconActionUPP

Creates a new universal procedure pointer (UPP) to an icon action callback function.

```
IconActionUPP NewIconActionUPP (
   IconActionProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your icon action function.

Return Value

A UPP to the icon action function.

Discussion

See the `IconActionProcPtr` (page 1302) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Icons.h`

NewIconGetterUPP

Creates a new universal procedure pointer (UPP) to an icon getter callback function.

```
IconGetterUPP NewIconGetterUPP (
   IconGetterProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your icon getter function.

Return Value

A UPP to the icon getter function.

Discussion

See the `IconGetterProcPtr` (page 1303) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Icons.h`

NewIconSuite

Gets a handle to an empty icon suite. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr NewIconSuite (
   IconSuiteRef *theIconSuite
);
```

Parameters

theIconSuite

On return, a pointer to a handle to a new, empty icon suite. Use the `AddIconToSuite` (page 1238) function to add handles to icon data.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

When you create an icon suite using this function, it sets the default label for the suite to none. To set a new default label for an icon suite, use the `SetSuiteLabel` (page 1300) function. `NewIconSuite` allocates the memory for the icon suite handle. To release the memory occupied by an icon suite, you must use the `DisposeIconSuite` (page 1241) function.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

OverrideIconRef

Replaces the bitmaps of one `IconRef` with those of another `IconRef`.

```
OSErr OverrideIconRef (
   IconRef oldIconRef,
   IconRef newIconRef
);
```

Parameters

oldIconRef

A pointer to a value of type `IconRef` whose bitmaps are to be replaced.

newIconRef

A pointer to a value of type`IconRef` containing the replacement bitmaps.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`IconsCore.h`

OverrideIconRefFromResource

Replaces the bitmaps in an `IconRef` with bitmaps from a specified resource file. (Deprecated in Mac OS X v10.5. Use `OverrideIconRef` (page 1274) instead.)

```
OSErr OverrideIconRefFromResource (
   IconRef theIconRef,
   const FSSpec *resourceFile,
   SInt16 resourceID
);
```

Parameters

theIconRef

An `IconRef` to be updated.

resourceFile

A pointer to the file system specification structure for the resource file containing the replacement bitmaps.

resourceID

The resource ID containing the replacement bitmaps. This value must be non-zero. You should provide a resource of type `'icns'` if possible. If an `'icns'` resource is not available, Icon Services uses standard icon suite resources, such as `'ICN#'`, instead.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`IconsCore.h`

PlotCIcon

Draws a color icon of resource type `'cicn'` to which you have a handle. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
void PlotCIcon (
   const Rect *theRect,
   CIconHandle theIcon
);
```

Parameters

theRect

A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

theIcon

A handle to the color icon structure of the color icon to draw. You can obtain a handle to the icon using the `GetCIcon` (page 1244) function, or `GetResource` or other Resource Manager functions.

Discussion

The `iconMask` field of the `CIcon` (page 1305) structure determines which pixels in the `iconPMap` field are drawn and which are not. Only pixels with 1s in corresponding positions in the `iconMask` field are drawn. If the screen depth is 1 or 2 bits per pixel, this function uses the `iconBMap` field instead of the `iconPMap` field (unless the `rowBytes` field of `IconBMap` contains 0, indicating that there is no bitmap for the icon).

When this function draws the icon, it uses the `bounds` field of `iconPMap` as the source rectangle of the image. If the destination rectangle is not the same size as the icon or its mask, the function stretches or shrinks the icon to fit. The icon's pixels are remapped to the current depth and color table, if necessary. The `bounds` fields of `iconPMap`, `iconBMap`, and `iconMask` are expected to be equal in size.

Unlike `PlotCIconHandle` (page 1276), this function does not allow you to specify any transforms or alignment. This function uses the QuickDraw function `CopyMask` and doesn't send any of its drawing commands through QuickDraw bottleneck functions. Therefore, calls to this function are not recorded as pictures.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

PlotCIconHandle

Draws an icon of resource type `'cicn'` to which you have a handle. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr PlotCIconHandle (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   CIconHandle theCIcon
);
```

Parameters

theRect

A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

align

Specifies how the function should align the icon within the rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

transform

Specifies how the function should modify the appearance of the icon. See "Icon Transformation Constants" (page 1309) for a description of the values you can use in this parameter.

theCIcon

A handle to the color icon structure of the icon to draw. You can obtain a handle to the icon using the `GetCIcon` (page 1244) function or `GetResource` or other Resource Manager functions.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Unlike `PlotCIcon` (page 1275), this function doesn't honor the current foreground and background colors.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

PlotIcon

Draws an icon of resource type `'ICON'` to which you have a handle. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
void PlotIcon (
   const Rect *theRect,
   Handle theIcon
);
```

Parameters

theRect

A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

theIcon

A handle to the icon to draw. You must have previously obtained this handle using the `GetIcon` (page 1246) function, or `GetResource` or other Resource Manager functions.

Discussion
This function does not allow you to specify any transforms or alignment. The `PlotIcon` function uses the QuickDraw function `CopyBits` with the `srcCopy` transfer mode. To plot an icon of resource type `'ICON'` with a specified transform and alignment, use the `PlotIconHandle` (page 1278) function.

If the destination rectangle is not 32 by 32 pixels, the function stretches or shrinks the icon to fit.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

PlotIconHandle

Draws an icon of resource type `'ICON'` or `'ICN#'` to which you have a handle. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr PlotIconHandle (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   Handle theIcon
);
```

Parameters

theRect
A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

align
Specifies how the function should align the icon within the rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

transform
Specifies how the function should modify the appearance of the icon. See "Icon Transformation Constants" (page 1309) for a description of the values you can use in this parameter.

theIcon
A handle to the icon to draw. You must have previously obtained a handle to the icon using the `GetIcon` (page 1246) function, or `GetResource` or other Resource Manager functions.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
To plot an icon from an icon suite, you should normally use `PlotIconSuite` (page 1282). This function may not draw the icon correctly if you pass it the handle returned in the `theIconData` parameter of `GetIconFromSuite` (page 1249).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

PlotIconID

Draws the icon described by an icon family. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr PlotIconID (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   SInt16 theResID
);
```

Parameters

theRect

A pointer to the rectangle, specified in local coordinates of the current graphics port, in which to draw the icon.

You cannot determine which icon from the family specified by `theResID` the function will draw. The function determines, from the size of the specified destination rectangle and the current bit depth of the display device, which icon of a given size to draw from an icon family. For example, if the destination rectangle has the coordinates (100,100,116,116) and the display device is set to 4-bit color, the function draws the icon of type `'ics4'` if that icon is available in the icon family.

If the width or height of a destination rectangle is greater than or equal to 32, the function uses the 32-by-32 pixel icon with the appropriate bit depth for the display device. If the destination rectangle is less than 32 by 32 pixels and greater than 16 pixels wide or 12 pixels high, `PlotIconID` uses the 16-by-16 pixel icon with the appropriate bit depth. If the destination rectangle's height is less than or equal to 12 pixels or its width is less than or equal to 16 pixels, `PlotIconID` uses the 12-by-16 pixel icon with the appropriate bit depth. (Typically only the Finder and Standard File Package use 12-by-16 pixel icons.)

The destination rectangle must be exactly 32 by 32 pixels, 16 by 16 pixels, or 12 by 16 pixels for the function to draw the icon without stretching it. If the destination rectangle is not one of these standard sizes, the function expands or shrinks the icon to fit.

align

Specifies how the function should align the icon within the rectangle. For example, you can specify that it center the icon within the rectangle or align it at one side or the other. The function moves the icon so that the edges of its mask align with the specified side or direction. See "Icon Alignment Constants" (page 1307) for a description of the values you can use here.

transform

Specifies how the function should modify the appearance of the icon. See "Icon Transformation Constants" (page 1309) for a description of the values you can use here.

theResID

The resource ID of the icon to draw. The icon resource must be of resource type `'ICN#'`, `'ics#'`, `'icl4'`, `'icl8'`, `'ics4'`, or `'ics8'`. In general, you should specify your icon resources as purgeable.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Icons.h`

PlotIconMethod

Draws an icon obtained with the aid of your icon getter callback function. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr PlotIconMethod (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   IconGetterUPP theMethod,
   void *yourDataPtr
);
```

Parameters

theRect

A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

align

Specifies how to align the icon within the specified rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use here.

transform

Specifies how the function should modify the appearance of the icon. See "Icon Transformation Constants" (page 1309) for a description of the values you can use here.

theMethod

A universal procedure pointer to your icon getter callback function. `PlotIconMethod` uses your icon getter function to obtain the icon to draw.

`PlotIconMethod` passes to your icon getter function the type of the icon to draw and the value specified in the `yourDataPtr` parameter. The `PlotIconMethod` function examines the current bit depth of the display devices and calls your icon getter function once for each display device that intersects the rectangle specified in the parameter `theRect`. Your icon getter function should return a handle to the requested icon's data. Your icon getter function can get the icon data using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons or use its icon getter function to get an icon from the desktop database.

For more information see the `IconGetterProcPtr` (page 1303) callback.

yourDataPtr

A pointer to data that is passed to your icon getter callback function.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

PlotIconRef

Draws an icon using appropriate size and depth data from an `IconRef`. (Deprecated in Mac OS X v10.5. Use `PlotIconRefInContext` (page 1281) instead.)

```
OSErr PlotIconRef (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   IconServicesUsageFlags theIconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

theRect
A pointer to the rectangle where the icon is to be drawn.

align
A value specifying how Icon Services should align the icon within the rectangle.

transform
A value specifying how Icon Services should modify the appearance of the icon.

theIconServicesUsageFlags
Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef
The `IconRef` for the icon to draw.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
This function is similar to the Icon Utilities function `PlotIconSuite`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

PlotIconRefInContext

Plots an `IconRef` using Quartz.

```
OSStatus PlotIconRefInContext (
   CGContextRef inContext,
   const CGRect *inRect,
   IconAlignmentType inAlign,
   IconTransformType inTransform,
   const RGBColor *inLabelColor,
   PlotIconRefFlags inFlags,
   IconRef inIconRef
);
```

Parameters

inContext
The graphics context to use.

inRect
A pointer to the rectangle to plot the icon in.

inAlign
The icon alignment. See "Icon Alignment Constants" (page 1307).

inTransform
The icon transform. See "Icon Transformation Constants" (page 1309).

inLabelColor
A pointer to the icon label color.

inFlags
The drawing flags to use; this is usually `kPlotIconRefNormalFlags`.

inIconRef
The `IconRef` to plot.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Icons.h`

PlotIconSuite

Draws the icon described by an icon suite using the most appropriate icon in the suite for the current bit depth of the display device and the rectangle in which the icon is to be drawn. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr PlotIconSuite (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   IconSuiteRef theIconSuite
);
```

Parameters

theRect

A pointer to the rectangle in which to draw the icon.

The function plots a single icon from the icon suite in the current graphics port. You cannot determine which icon from a given suite it will draw; the function bases this decision on the size of the specified destination rectangle and the current bit depth of the display device. For example, if the destination rectangle has the coordinates (100,100,116,116) and the display device is set to 4-bit color, the function draws the icon of type `'ics4'` if that icon is available in the icon suite.

If the width or height of a destination rectangle is greater than or equal to 32 pixels, the function uses the 32-by-32 pixel icon with the appropriate bit depth for the display device. If the destination rectangle is less than 32 by 32 pixels and greater than 16 pixels wide or 12 pixels high, the function uses the 16-by-16 pixel icon with the appropriate bit depth. If the destination rectangle's height is less than or equal to 12 pixels or its width is less than or equal to 16 pixels, the function uses the 12-by-16 pixel icon with the appropriate bit depth. (Typically, only the Finder and Standard File Package use 12-by-16 pixel icons.)

The destination rectangle passed in the `theRect` parameter must be exactly 32 by 32 pixels, 16 by 16 pixels, or 12 by 16 pixels for the function to draw the icon without stretching it. If the destination rectangle is not one of these standard sizes, the function expands or shrinks the icon to fit.

align

Specifies how the function should align the icon within the rectangle. For example, you can specify that the function center the icon within the rectangle or align it at one side or the other. See "Icon Alignment Constants" (page 1307) for a description of the values you can use here.

transform

Specifies how the function should modify the appearance of the icon. See "Icon Transformation Constants" (page 1309) for a description of the values you can use here.

If you don't specify a label constant in this parameter, the function displays the icon using the default label for that icon suite. When you create an icon suite using the `GetIconSuite` (page 1257) function or the `NewIconSuite` (page 1273) function, these functions set the default label for the suite to none. To set a new default label for an icon suite, use the `SetSuiteLabel` (page 1300) function.

theIconSuite

A handle to the icon suite from which the function gets the icon to draw. You can get a handle to an icon suite using the `GetIconSuite` or `NewIconSuite` functions.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Icons.h`

PlotSICNHandle

Draws a small icon of resource type `'SICN'` to which you have a handle. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr PlotSICNHandle (
   const Rect *theRect,
   IconAlignmentType align,
   IconTransformType transform,
   Handle theSICN
);
```

Parameters

theRect
A pointer to the rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

align
Specifies how the function should align the icon within the rectangle. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

transform
Specifies how the function should modify the appearance of the icon. See "Icon Transformation Constants" (page 1309) for a description of the values you can use in this parameter.

theSICN
A handle to the icon to draw. You can obtain a handle to the icon using `GetResource` or other Resource Manager functions.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Only `'SICN'` resources with a single member—or with two members, the second of which is a mask for the first—plot correctly.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

PtInIconID

Determines whether a specified point is within an icon. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
Boolean PtInIconID (
   Point testPt,
   const Rect *iconRect,
   IconAlignmentType align,
   SInt16 iconID
);
```

Parameters

testPt

The point to be tested, specified in local coordinates of the current graphics port. A point is considered to be within an icon if the point is within the icon's mask.

iconRect

A pointer to the rectangle in which the icon appears, specified in local coordinates of the current graphics port. The function determines, from the size of the rectangle specified in this parameter, which icon mask from the given icon family to test the point against. The rectangle which you specify here should be the same rectangle that you last used to draw the icon. The function then uses the location of this rectangle (and the alignment of the icon in the rectangle) to determine whether the specified point is within the icon.

align

Specifies how the icon against which to hit-test is aligned within the rectangle specified by the `iconRect` parameter. The alignment which you specify here should be the same alignment that you last used to draw the icon. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

iconID

A resource ID for an icon family. In general, you should specify your icon resources as purgeable.

Return Value

`TRUE` if the point is in the icon and `FALSE` if it is not.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

PtInIconMethod

Determines whether a specified point is within an icon obtained with the aid of your icon getter callback function. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
Boolean PtInIconMethod (
   Point testPt,
   const Rect *iconRect,
   IconAlignmentType align,
   IconGetterUPP theMethod,
   void *yourDataPtr
);
```

Parameters

testPt

The point to be tested, specified in local coordinates of the current graphics port. A point is considered to be within an icon if the point is within the icon's mask.

iconRect

A pointer to the rectangle in which the icon appears, specified in local coordinates of the current graphics port. The rectangle which you specify here should be the same rectangle that you last used to draw the icon.

align

Specifies how the icon against which to hit-test is aligned within the rectangle specified by the `iconRect` parameter. The alignment which you specify here should be the same alignment that you last used to draw the icon. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

theMethod

A universal procedure pointer to your icon getter callback function. `PtInIconMethod` passes to your icon getter function the type of icon your function should retrieve (either `'ICN#'` or `'ics#'`) and also passes the value specified in the `yourDataPtr` parameter. The `PtInIconMethod` function examines the size of the specified rectangle and requests the appropriate icon from your icon getter function. Your icon getter function should return a handle to the requested icon's data. The `PtInIconMethod` function extracts the mask from the icon data that your icon getter function returns. If your icon getter function returns data that does not correspond to an icon of type `'ICN#'` or type `'ics#'`, `PtInIconMethod` attempts to generate a mask from the returned data.

Your icon getter function can get the icon's data using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons (and pass a pointer to it in the `yourDataPtr` parameter) or use its icon getter function to get an icon from the desktop database.

See the `IconGetterProcPtr` (page 1303) callback for more information on creating an icon getter function.

yourDataPtr

A pointer to data that is passed to your icon getter function.

Return Value

`TRUE` if the point is in the icon and `FALSE` if it is not.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Icons.h`

PtInIconRef

Tests whether a specified point falls within an icon's mask. (Deprecated in Mac OS X v10.5. Use `IconRefContainsCGPoint` (page 1262) instead.)

```
Boolean PtInIconRef (
   const Point *testPt,
   const Rect *iconRect,
   IconAlignmentType align,
   IconServicesUsageFlags theIconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

testPt
A pointer to the location, specified in local coordinates of the current graphics port, that Icon Services tests to see whether it falls within the mask of the indicated icon.

iconRect
A pointer to the rectangle defining the area that Icon Services uses to determine which icon is hit-tested. Use the same `Rect` value as when the icon was last drawn.

align
A value that specifies how the indicated icon is aligned within the rectangle specified in the `iconRect` parameter. Use the same `IconAlignmentType` value as when the icon was last drawn. for a description of possible return values, see "Icon Alignment Constants" (page 1307).

theIconServicesUsageFlags
Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef
The icon to be tested.

Return Value
`true` if the point specified in the `testPt` parameter falls within the appropriate icon mask, `false` otherwise.

Discussion
This function is similar to the Icon Utilities function `PtInIconSuite`. The function is useful when you want to determine whether a user has clicked on a particular icon, for example.

Icon Services uses the icon's black-and-white mask for hit-testing, even if you provide a deep mask.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

PtInIconSuite

Determines whether a specified point is within an icon. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
Boolean PtInIconSuite (
   Point testPt,
   const Rect *iconRect,
   IconAlignmentType align,
   IconSuiteRef theIconSuite
);
```

Parameters

testPt

The point to be tested, specified in local coordinates of the current graphics port. A point is considered to be within an icon if the point is within the icon's mask.

iconRect

A pointer to the rectangle in which the icon appears, specified in local coordinates of the current graphics port. The function determines, from the size of the rectangle specified in this parameter, which icon mask (`'ICN#'` or `'ics#'`) from the specified icon suite to test the point against. The function then uses the location of this rectangle (and the location of the icon in the rectangle) to determine whether the given point is within the icon. The rectangle which you specify here should be the same rectangle that you last used to draw the icon.

align

Specifies how the icon against which to hit-test is aligned within the rectangle specified by the `iconRect` parameter. The alignment which you specify here should be the same alignment that you last used to draw the icon. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

theIconSuite

A handle to an icon suite.

Return Value

`TRUE` if the point is in the icon and `FALSE` if it is not.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

ReadIconFile

Copies data from a given file into an icon family. (Deprecated in Mac OS X v10.5. Use `ReadIconFromFSRef` (page 1289) instead.)

```
OSErr ReadIconFile (
   const FSSpec *iconFile,
   IconFamilyHandle *iconFamily
);
```

Parameters

iconFile

A pointer to the file specification structure for the source file for icon data.

iconFamily

A handle to an `iconFamily` data structure to be used as the target data structure. Icon Services resizes the handle as needed. For more information on the `IconFamily` data structure, see 'icns'.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`IconsCore.h`

ReadIconFromFSRef

Reads an icon ('`icns`') file into memory.

```
OSStatus ReadIconFromFSRef (
   const FSRef *ref,
   IconFamilyHandle *iconFamily
);
```

Parameters

ref

A pointer to the `FSRef` for the icon file.

iconFamily

A pointer to the handle for the icon family.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`IconsCore.h`

RectInIconID

Hit-tests a rectangle against the appropriate icon mask from an icon family for a specified destination rectangle and alignment. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
Boolean RectInIconID (
   const Rect *testRect,
   const Rect *iconRect,
   IconAlignmentType align,
   SInt16 iconID
);
```

Parameters

testRect

A pointer to the rectangle to be tested, specified in local coordinates of the current graphics port.

iconRect

A pointer to the rectangle in which the icon appears, specified in local coordinates of the current graphics port. The rectangle which you specify here should be the same rectangle that you last used to draw the icon. Like the `PtInIconID` (page 1285) function, this function determines, from the size of the rectangle specified in this parameter, which icon mask from the icon family to test the `testRect` parameter against.

align

Specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. The alignment which you specify here should be the same alignment that you last used to draw the icon. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

iconID

A resource ID for an icon family. In general, you should specify your icon resources as purgeable.

Return Value

`TRUE` if the rectangle intersects the icon and `FALSE` if it doesn't.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

RectInIconMethod

Hit-tests a rectangle against an icon obtained by your icon getter callback function for a specified destination rectangle and alignment. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
Boolean RectInIconMethod (
   const Rect *testRect,
   const Rect *iconRect,
   IconAlignmentType align,
   IconGetterUPP theMethod,
   void *yourDataPtr
);
```

Parameters

testRect

A pointer to the rectangle to be tested, specified in local coordinates of the current graphics port.

iconRect

A pointer to the rectangle in which the icon appears, specified in local coordinates of the current graphics port. The rectangle which you specify here should be the same rectangle that you last used to draw the icon.

align

Specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. The alignment which you specify here should be the same alignment that you last used to draw the icon. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

theMethod

A universal procedure pointer to your icon getter callback function. `RectInIconMethod` passes to your icon getter function the type of the icon your function should retrieve and the value specified in the `yourDataPtr` parameter. The `RectInIconMethod` function examines the size of the rectangle and requests the appropriate icon from your icon getter function—an icon of icon type `'ICN#'` or `'ics#'`. Your icon getter function should return a handle to the data of the requested icon type. The `RectInIconMethod` function extracts the mask from the icon data that your icon getter function returns. If your icon getter function returns data that does not correspond to an icon of type `'ICN#'` or type `'ics#'`, `RectInIconMethod` attempts to generate a mask from the returned data.

Your icon getter function can get the data for the icon and its mask using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons (and pass a pointer to it in the `yourDataPtr` parameter) or use its icon getter function to get an icon from the desktop database.

See the `IconGetterProcPtr` (page 1303) callback for more information on creating an icon getter function.

yourDataPtr

A pointer to data that is passed to your icon getter function.

Return Value

`TRUE` if the rectangle intersects the icon and `FALSE` if it doesn't.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

RectInIconRef

Tests whether a specified rectangle falls within an icon's mask. (Deprecated in Mac OS X v10.5. Use `IconRefIntersectsCGRect` (page 1263) instead.)

```
Boolean RectInIconRef (
   const Rect *testRect,
   const Rect *iconRect,
   IconAlignmentType align,
   IconServicesUsageFlags iconServicesUsageFlags,
   IconRef theIconRef
);
```

Parameters

testRect

A pointer to the rectangle, specified in local coordinates of the current graphics port, that Icon Services tests to see whether it falls within the mask of the indicated icon.

iconRect

A pointer to the area that Icon Services uses to determine which icon is hit-tested. Use the same `Rect` value as when the icon was last drawn.

align

A value that specifies how the indicated icon is aligned within the rectangle specified in the `iconRect` parameter. Use the same `IconAlignmentType` value as when the icon was last drawn. for a description of possible return values, see "Icon Alignment Constants" (page 1307).

iconServicesUsageFlags

Reserved for future use. Pass the `kIconServicesNormalUsageFlag` constant in this parameter.

theIconRef

A pointer to a value of type `IconRef` specifying the icon family to use for drawing the requested icon.

Return Value

`true` if the rectangle specified in the `testRect` parameter intersects the appropriate icon mask, `false` otherwise.

Discussion

This function is similar to the Icon Utilities function `RectInIconSuite`. The function is useful when you want to determine whether a user selection intersects a particular icon, for example.

Icon Services uses the icon's black-and-white mask for hit-testing, even if you provide a deep mask.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

RectInIconSuite

Hit-tests a rectangle against the appropriate icon mask from an icon suite for a specified destination rectangle and alignment. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
Boolean RectInIconSuite (
   const Rect *testRect,
   const Rect *iconRect,
   IconAlignmentType align,
   IconSuiteRef theIconSuite
);
```

Parameters

testRect

A pointer to the rectangle to be tested, specified in local coordinates of the current graphics port.

iconRect

A pointer to the rectangle in which the icon appears, specified in local coordinates of the current graphics port. The rectangle which you specify here should be the same rectangle that you last used to draw the icon. Like the `PtInIconSuite` (page 1288) function, this function determines, from the size of the rectangle specified in this parameter, which icon mask from the icon suite specified by `theIconSuite` to test the test rectangle against. For example, if the coordinates of the `iconRect` parameter are (100,100,116,116) and the icon cache contains entries for each icon family member, `RectInIconSuite` uses the icon mask defined by the `'ics#'` entry.

The function then intersects the rectangle specified by `testRect` with the icon mask in the `iconRect` rectangle.

align

Specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. The alignment which you specify here should be the same alignment that you last used to draw the icon. See "Icon Alignment Constants" (page 1307) for a description of the values you can use in this parameter.

theIconSuite

A handle to an icon suite.

Return Value

`TRUE` if the rectangle intersects the icon and `FALSE` if it doesn't.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

RegisterIconRefFromFSRef

Registers an `IconRef` from a `.icns` file and associates it with a creator and type pair.

```
OSStatus RegisterIconRefFromFSRef (
   OSType creator,
   OSType iconType,
   const FSRef *iconFile,
   IconRef *theIconRef
);
```

Parameters

creator

The creator code for the `.icns` file.

iconType

The type code for the `.icns` file.

iconFile

A pointer to the `FSRef` of the `.icns` file.

theIconRef

A pointer to an `IconRef`. On return, this contains the registered `IconRef`.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`IconsCore.h`

RegisterIconRefFromIconFamily

Adds an `iconFamily`-derived `IconRef` to the Icon Services registry.

```
OSErr RegisterIconRefFromIconFamily (
   OSType creator,
   OSType iconType,
   IconFamilyHandle iconFamily,
   IconRef *theIconRef
);
```

Parameters

creator

The creator code of the desired icon. You can use your application's creator code, for example. Lower-case creator codes are reserved for the System.

iconType

The type code of the desired icon.

iconFamily

A handle to the `iconFamily` data structure to register.

theIconRef

On return, a pointer to the desired icon data.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
Consider using the function `RegisterIconRefFromIconFile` (page 1295), since the data registered using the `RegisterIconRefFromIconFamily` function cannot be purged. You are responsible for disposing of the `IconRef` by using the function `ReleaseIconRef` (page 1296).

Calling this function increments the reference count of the `IconRef`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

RegisterIconRefFromIconFile

Adds a file-derived `IconRef` to the Icon Services registry. (Deprecated in Mac OS X v10.5. Use `RegisterIconRefFromFSRef` (page 1293) instead.)

```
OSErr RegisterIconRefFromIconFile (
   OSType creator,
   OSType iconType,
   const FSSpec *iconFile,
   IconRef *theIconRef
);
```

Parameters

creator
The creator code of the icon data you wish to register. You can use your application's creator code, for example. Lower-case creator codes are reserved for the system.

iconType
The type code of the icon data you wish to register.

iconFile
A pointer to the file system specification structure for the file to use as the icon data source.

theIconRef
On return, a pointer to the desired icon data.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`IconsCore.h`

RegisterIconRefFromResource

Adds a resource-derived `IconRef` to the Icon Services registry. (Deprecated in Mac OS X v10.5. Use `RegisterIconRefFromFSRef` (page 1293) instead.)

```
OSErr RegisterIconRefFromResource (
   OSType creator,
   OSType iconType,
   const FSSpec *resourceFile,
   SInt16 resourceID,
   IconRef *theIconRef
);
```

Parameters

creator

The creator code of the icon data you wish to register. You can use your application's creator code, for example. Lower-case creator codes are reserved for the system.

iconType

The type code of the icon data you wish to register.

resourceFile

A pointer to the file system specification structure for the resource file from which to read the icon data.

resourceID

The resource ID of the icon data to be registered. This value must be non-zero.

You should provide a resource of type `'icns'` if possible. If an `'icns'` resource is not available, Icon Services uses standard icon suite resources, such as `'ICN#'`, instead.

theIconRef

On return, a pointer to the desired icon data.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

You can use the `RegisterIconRefFromResource` function to register icons from `'icns'` resources or "classic" custom icon resources (`'ics#'` , `'ICN#'` , etc.). Icon Services searches `'icns'` resources before searching other icon resources.

Calling this function increments the reference count of the `IconRef`.

Remember to call the function `ReleaseIconRef` (page 1296) when you're done with an `IconRef`.

Special Considerations

Before using the recommended replacement function, you need to move the contents of the icon resource into an icon family `.icns` file.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`IconsCore.h`

ReleaseIconRef

Decrements the reference count for an `IconRef`.

```
OSErr ReleaseIconRef (
   IconRef theIconRef
);
```

Parameters

theIconRef

An `IconRef` whose reference count you wish to decrement.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

When an `IconRef`'s reference count reaches 0, all memory allocated for the `IconRef` is marked as disposable. Any subsequent attempt to use the `IconRef` returns a result code of - `2580` (`invalidIconRefErr`).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`IconsCore.h`

RemoveIconRefOverride

Restores the original bitmaps of an overridden `IconRef`.

```
OSErr RemoveIconRefOverride (
   IconRef theIconRef
);
```

Parameters

theIconRef

A pointer to a value of type `IconRef` whose bitmaps are to be restored.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`IconsCore.h`

SetCustomIconsEnabled

Enables or disables custom icons on a specified volume.

```
OSErr SetCustomIconsEnabled (
   SInt16 vRefNum,
   Boolean enableCustomIcons
);
```

Parameters

vRefNum

The volume where custom icons are to be enabled or disabled.

enableCustomIcons
If you pass `true`, custom icons are enabled on the volume specified. Passing `false` disables custom icons on the volume specified.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
If you use the `SetCustomIconsEnabled` function to enable or disable custom icons, the setting remains in effect only as long as the specified volume remains mounted during the current session.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

SetIconCacheData

Sets the data associated with an icon cache. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr SetIconCacheData (
   IconCacheRef theCache,
   void *theData
);
```

Parameters
theCache
A reference to the icon cache whose data is to be set.

theData
A pointer to the data to set.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
All the Icon Utilities functions that accept a handle to an icon suite also accept a handle to an icon cache.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

SetIconCacheProc

Sets the icon getter callback function associated with an icon cache. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr SetIconCacheProc (
   IconCacheRef theCache,
   IconGetterUPP theProc
);
```

Parameters

theCache

A reference to the icon cache whose icon getter function is to be set.

theProc

A universal procedure pointer to the icon getter callback function to associate with the specified cache. See the `IconGetterProcPtr` (page 1303) callback for more information on icon getter functions.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

All the Icon Utilities functions that accept a handle to an icon suite also accept a handle to an icon cache. An icon cache is like an icon suite except that it also contains a pointer to an icon getter callback function and a pointer to data that can be used as a reference constant. An icon cache typically does not contain handles to the icon resources for all icon family members. Instead, if the icon cache does not contain an entry for a specific type of icon in an icon family, the Icon Utilities functions call your application's icon getter function to retrieve the data for that icon type.

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Icons.h`

SetIconFamilyData

Provides new raw icon data for an individual element of an icon family.

```
OSErr SetIconFamilyData (
   IconFamilyHandle iconFamily,
   OSType iconType,
   Handle h
);
```

Parameters

iconFamily

A handle to an `iconFamily` data structure to be used as the target.

iconType

The format of the icon data you provide.You may specify one of the icon types (as defined in `IconStorage.h` in the CoreServices/OSServices framework) or `'PICT'` in this parameter. For a thumbnail icon, for example, you specify `kThumbnail32BitData` in this parameter. For a thumbnail mask, you specify `kThumbnail8BitMask`.

h

A handle to the icon data you provide. For a thumbnail icon, the handle contains raw image data in the form of 128x128, four bytes per pixel, RGB data. For a thumbnail mask, the data is in the same format except that it is one byte per pixel.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

SetSuiteLabel

Specifies the default label associated with an icon suite. (Deprecated in Mac OS X v10.5. Use Icon Services instead.)

```
OSErr SetSuiteLabel (
   IconSuiteRef theSuite,
   SInt16 theLabel
);
```

Parameters

theSuite

A handle to an icon suite.

theLabel

An integer from 1 to 7 that specifies a label for the icon suite, or 0 to set the icon suite's label to none. The default label setting helps to determine which of the label colors shown in the Finder's Label menu is applied to icons of that suite when your application displays them.

You can override the default label setting for a suite by specifying a label in the `transform` parameter of the `PlotIconSuite` (page 1282) function. For example, suppose the color currently set for the third label displayed in the Finder's Label menu is red, and the color for the fourth label is green. If you set the default label for a suite using `SetSuiteLabel(theSuite,3)`, then draw an icon from the same suite using `PlotIconSuite` and specifying `kTransformNone` in the `transform` parameter, the label color red is applied to the icon. However, if you specify `kTransformLabel4` in the `transform` parameter of the `PlotIconSuite` function, the label color green is applied to the icon.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

This function may move or purge memory blocks in the application heap. For that reason, your application should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Icons.h`

UnregisterIconRef

Removes the specified icon data from the icon registry.

```
OSErr UnregisterIconRef (
   OSType creator,
   OSType iconType
);
```

Parameters

creator
The creator code of the icon data to be unregistered.

iconType
The type code of the icon data to be unregistered.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
The specified icon data is not unregistered until all its users have called the function `ReleaseIconRef` (page 1296).

You should not unregister an icon that you have not registered.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

UpdateIconRef

Forces an update of `IconRef` data.

```
OSErr UpdateIconRef (
   IconRef theIconRef
);
```

Parameters

theIconRef
An `IconRef` to be updated.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion
This function is useful after you have changed a file or folder's custom icon, for example. Do not call the `UpdateIconRef` function if you have not already obtained an `IconRef` for a particular icon; call the function `GetIconRefFromFile` (page 1251) instead.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

WriteIconFile

Copies data from a given icon family into a file. (Deprecated in Mac OS X v10.5. Use the File Manager instead.)

```
OSErr WriteIconFile (
   IconFamilyHandle iconFamily,
   const FSSpec *iconFile
);
```

Parameters

iconFamily
A handle to an `iconFamily` data structure to be used as a source for icon data. For more information on the `IconFamily` data structure, see 'icns'.

iconFile
A pointer to the file specification structure for the file to use as the target for icon data.

Return Value
A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Special Considerations

Icon Services is designed to read icon data from a file and cache the data, but not to write out icon data. You can use File Manager functions to write your icon data to a file.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`IconsCore.h`

Callbacks

IconActionProcPtr

Defines a pointer to an icon action callback function, which performs an action on a single icon.

```
typedef OSErr (*IconActionProcPtr) (
    ResType theType,
    Handle *theIcon,
    void *yourDataPtr
);
```

If you name your function `MyIconActionProc`, you would declare it like this:

```
OSErr MyIconActionProc (
    ResType theType,
    Handle *theIcon,
    void *yourDataPtr
);
```

Parameters

theType

The resource type of the icon.

theIcon

A pointer to the handle to the icon on which to perform the operation.

yourDataPtr

A pointer to data as specified in the `yourDataPtr` parameter of the `ForEachIconDo` function. When your application calls `ForEachIconDo`, it typically provides in the `yourDataPtr` parameter a value that identifies the action your function should perform.

Return Value

A result code. See "Icon Services and Utilities Result Codes" (page 1324).

Discussion

You can perform operations on every icon in an icon suite by providing a pointer to an icon action function as a parameter to the `ForEachIconDo` (page 1244) function. The `ForEachIconDo` function calls your icon action function for specified icon resource types. Your icon action function should return a result code indicating whether it successfully performed the action on the icon.

Before using your icon action function, you must first create a new universal procedure pointer to it, using the `NewIconActionUPP` (page 1272) function, as shown here:

```
IconActionUPP MyIconActionUPP;
MyIconActionUPP = NewIconActionUPP(&MyIconActionProc)
```

You then pass `MyIconActionUPP` to the `ForEachIconDo` function. When you are finished with your icon action callback function, you should dispose of the universal procedure pointer associated with it:

```
DisposeIconActionUPP(MyIconActionUPP);
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Icons.h`

IconGetterProcPtr

Defines a pointer to an icon getter callback function, which retrieves a handle to an icon of the requested type.

```
typedef Handle (*IconGetterProcPtr) (
    ResType theType,
    void *yourDataPtr
);
```

If you name your function `MyIconGetterProc`, you would declare it like this:

```
Handle MyIconGetterProc (
    ResType theType,
    void *yourDataPtr
);
```

Parameters

theType

The resource type of the icon. In general, you should specify your icon resources as purgeable.

yourDataPtr

If your icon getter function was called by an icon cache function, this parameter contains, on return, a pointer to the data associated with the icon cache. Otherwise, this parameter contains the value your application specified in the `yourDataPtr` parameter. For icon caches, you initially set this value when you first create a cache using the `MakeIconCache` (page 1272) function. You can change this value using the `SetIconCacheData` (page 1298) function. The icon getter function can use this data as needed.

Return Value

An icon getter function should return as its function result a handle to the requested icon's data.

Discussion

If you use icon caches, you must provide at least one icon getter function. The `MakeIconCache` function takes a pointer to an icon getter function for use with a new icon cache. Subsequent calls to Icon Utilities functions that use icon types not present in the icon cache use the icon getter function associated with the icon cache to return a handle to the icon data. To get and set an existing icon cache's icon getter function, use the `GetIconCacheProc` (page 1247) and `SetIconCacheProc` (page 1299) functions.

You can also specify an icon getter function for use by the `PlotIconMethod` (page 1280), `IconMethodToRgn` (page 1261), `PtInIconMethod` (page 1285), and `RectInIconMethod` (page 1290) functions. Like Icon Utilities functions that work with icon caches, the icon getter function that you provide as a parameter to `PlotIconMethod` should return a handle to the requested icon's data. Note that the icon getter function that you provide as a parameter to `IconMethodToRgn`, `PtInIconMethod`, and `RectInIconMethod` should also return a handle to the requested icon; these three functions then extract the icon mask from the icon data your icon getter function returns.

Before using your icon getter function, you must first create a new universal procedure pointer to it, using the `NewIconGetterUPP` (page 1273) function, as shown here:

```
IconGetterUPP MyIconGetterUPP;
MyIconGetterUPP = NewIconGetterUPP(&MyIconGetterProc)
```

You can then pass `MyIconGetterUPP` to any of the Icon Utilities functions which use custom icon getter functions. When you are finished with your icon getter callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeIconGetterUPP` (page 1241) function:

```
DisposeIconGetterUPP(MyIconGetterUPP);
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

Data Types

CIcon

Defines a color icon structure.

```
struct CIcon {
    PixMap iconPMap;
    BitMap iconMask;
    BitMap iconBMap;
    Handle iconData;
    SInt16 iconMaskData[1];
};
typedef struct CIcon CIcon;
typedef CIcon * CIconPtr;
```

Fields

`iconPMap`
The pixel map describing the icon. Note that this is a pixel map record, not a handle to a pixel map record.

`iconMask`
A bitmap of the icon's mask.

`iconBMap`
A bitmap of the icon.

`iconData`
A handle to the icon's pixel image.

`iconMaskData`
An array containing the icon's mask data followed by the icon's bitmap data. This is used only when the icon is stored as a resource.

Discussion
The `PlotCIcon` (page 1275), `PlotCIconHandle` (page 1276), `GetCIcon` (page 1244), and `DisposeCIcon` (page 1240) functions all use the `CIconHandle` data type to refer to a color icon structure. A color icon structure contains information about a color icon.

All color icon resources should be marked purgeable. You can use icons of resource type `'cicn'` in menus the same way that you use resources of type `'ICON'`. If a menu item specifies an icon number, the menu definition function first tries to load in a `'cicn'` resource with the specified resource ID. If it doesn't find one, the menu definition function tries to load in an `'ICON'` resource with the same ID. The Dialog Manager also uses a `'cicn'` resource instead of an `'ICON'` resource if it finds one with the same resource ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Icons.h`

IconRef

Defines an icon reference.

```
typedef struct OpaqueIconRef * IconRef;
```

Discussion
An `IconRef` is a 32–bit values identifying cached icon data. `IconRef` 0 is invalid.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IconsCore.h`

IconActionUPP

Defines a universal procedure pointer (UPP) to an icon action callback function.

```
typedef IconActionProcPtr IconActionUPP;
```

Discussion
For more information, see the description of the `IconActionProcPtr` (page 1302) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

IconGetterUPP

Defines a universal procedure pointer to an icon getter callback function.

```
typedef IconGetterProcPtr IconGetterUPP;
```

Discussion
For more information, see the description of the `IconGetterProcPtr` (page 1303) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Icons.h`

IconCacheRef

Defines a reference to an icon cache.

```
typedef Handle IconCacheRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Icons.h`

IconSuiteRef

Defines a reference to an icon suite.

```
typedef Handle IconSuiteRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Icons.h`

Constants

Icon Alignment Constants

Define constants that allow you to specify how to align an icon within its rectangle.

```
enum {
    kAlignNone = 0x00,
    kAlignVerticalCenter = 0x01,
    kAlignTop = 0x02,
    kAlignBottom = 0x03,
    kAlignHorizontalCenter = 0x04,
    kAlignAbsoluteCenter = kAlignVerticalCenter | kAlignHorizontalCenter,
    kAlignCenterTop = kAlignTop | kAlignHorizontalCenter,
    kAlignCenterBottom = kAlignBottom | kAlignHorizontalCenter,
    kAlignLeft = 0x08,
    kAlignCenterLeft = kAlignVerticalCenter | kAlignLeft,
    kAlignTopLeft = kAlignTop | kAlignLeft,
    kAlignBottomLeft = kAlignBottom | kAlignLeft,
    kAlignRight = 0x0C,
    kAlignCenterRight = kAlignVerticalCenter | kAlignRight,
    kAlignTopRight = kAlignTop | kAlignRight,
    kAlignBottomRight = kAlignBottom | kAlignRight
};
typedef SInt16 IconAlignmentType;
```

Constants
`kAlignNone`
Use this value if you do not wish to specify a particular alignment.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignVerticalCenter`
Use this value to center the icon vertically within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignTop`
Use this value to top align the icon within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignBottom`
Use this value to bottom align the icon within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignHorizontalCenter`
Use this value to center the icon horizontally within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignAbsoluteCenter`
Use this value to center the icon horizontally and vertically within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignCenterTop`
Use this value to top align the icon and center it horizontally within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignCenterBottom`
Use this value to bottom align the icon and center it horizontally within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignLeft`
Use this value to left align the icon within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignCenterLeft`
Use this value to left align the icon and center it vertically within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignTopLeft`
Use this value to left and top align the icon within the rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kAlignBottomLeft`
Use this value to left and bottom align the icon within the rectangle.

Available in Mac OS X v10.0 and later.

Declared in `Icons.h`.

`kAlignRight`
Use this value to right align the icon within the rectangle.

Available in Mac OS X v10.0 and later.

Declared in `Icons.h`.

`kAlignCenterRight`
Use this value to right align the icon and center it vertically within the rectangle.

Available in Mac OS X v10.0 and later.

Declared in `Icons.h`.

`kAlignTopRight`
Use this value to right and top align the icon within the rectangle.

Available in Mac OS X v10.0 and later.

Declared in `Icons.h`.

`kAlignBottomRight`
Use this value to right and bottom align the icon within the rectangle.

Available in Mac OS X v10.0 and later.

Declared in `Icons.h`.

Discussion
Icon Services and Utilities functions use the `IconAlignmentType` constants to determine how an icon is aligned within its bounding rectangle.

Icon Transformation Constants

Define values that Icon Services uses to report how an icon has been transformed after you call the function `GetIconRefVariant`.

```
enum {
    kTransformNone = 0x00,
    kTransformDisabled = 0x01,
    kTransformOffline = 0x02,
    kTransformOpen = 0x03,
    kTransformLabel1 = 0x0100,
    kTransformLabel2 = 0x0200,
    kTransformLabel3 = 0x0300,
    kTransformLabel4 = 0x0400,
    kTransformLabel5 = 0x0500,
    kTransformLabel6 = 0x0600,
    kTransformLabel7 = 0x0700,
    kTransformSelected = 0x4000,
    kTransformSelectedDisabled = kTransformSelected | kTransformDisabled,
    kTransformSelectedOffline = kTransformSelected | kTransformOffline,
    kTransformSelectedOpen = kTransformSelected | kTransformOpen
};
typedef SInt16 IconTransformType;
```

Discussion
The functions `PlotIconID` (page 1279)`PlotIconMethod` (page 1280), `PlotIconHandle` (page 1278), `PlotCIconHandle` (page 1276), `PlotIconSuite` (page 1282), `LoadIconCache` (page 1270) and `PlotSICNHandle` (page 1284) use these constants to specify how an icon should be modified, if at all, when plotted.

Icon Selector Constants

Describe values that you can use to obtain information about the sizes and depths of icons available in a given icon family.

```
enum {
    kSelectorLarge1Bit = 0x00000001,
    kSelectorLarge4Bit = 0x00000002,
    kSelectorLarge8Bit = 0x00000004,
    kSelectorLarge32Bit = 0x00000008,
    kSelectorLarge8BitMask = 0x00000010,
    kSelectorSmall1Bit = 0x00000100,
    kSelectorSmall4Bit = 0x00000200,
    kSelectorSmall8Bit = 0x00000400,
    kSelectorSmall32Bit = 0x00000800,
    kSelectorSmall8BitMask = 0x00001000,
    kSelectorMini1Bit = 0x00010000,
    kSelectorMini4Bit = 0x00020000,
    kSelectorMini8Bit = 0x00040000,
    kSelectorHuge1Bit = 0x01000000,
    kSelectorHuge4Bit = 0x02000000,
    kSelectorHuge8Bit = 0x04000000,
    kSelectorHuge32Bit = 0x08000000,
    kSelectorHuge8BitMask = 0x10000000,
    kSelectorAllLargeData = 0x000000FF,
    kSelectorAllSmallData = 0x0000FF00,
    kSelectorAllMiniData = 0x00FF0000,
    kSelectorAllHugeData = 0xFF000000,
    kSelectorAll1BitData = kSelectorLarge1Bit | kSelectorSmall1Bit
| kSelectorMini1Bit | kSelectorHuge1Bit,
    kSelectorAll4BitData = kSelectorLarge4Bit | kSelectorSmall4Bit
| kSelectorMini4Bit | kSelectorHuge4Bit,
    kSelectorAll8BitData = kSelectorLarge8Bit | kSelectorSmall8Bit
| kSelectorMini8Bit | kSelectorHuge8Bit,
    kSelectorAll32BitData = kSelectorLarge32Bit | kSelectorSmall32Bit
| kSelectorHuge32Bit,
    kSelectorAllAvailableData = 0xFFFFFFFF
};
typedef UInt32 IconSelectorValue;
```

Constants

`kSelectorLarge1Bit`
Specify to include `'ICN#'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorLarge4Bit`
Specify to include `'icl4'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorLarge8Bit`
Specify to include `'icl8'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorLarge32Bit`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorLarge8BitMask`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorSmall1Bit`
Specify to include `'ics#'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorSmall4Bit`
Specify to include `'ics4'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorSmall8Bit`
Specify to include `'ics8'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorSmall32Bit`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorSmall8BitMask`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorMini1Bit`
Specify to include `'icm#'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorMini4Bit`
Specify to include `'icm4'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorMini8Bit`
Specify to include `'icm8'` resource.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorHuge1Bit`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorHuge4Bit`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorHuge8Bit`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorHuge32Bit`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorHuge8BitMask`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAllLargeData`
Specify to include `'ICN#'`, `'icl4'`, and `'icl8'` resources.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAllSmallData`
Specify to include `'ics#'`, `'ics4'`, and `'ics8'` resources.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAllMiniData`
Specify to include `'icm#'`, `'icm4'`, and `'icm8'` resources.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAllHugeData`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAll1BitData`
Specify to include `'ICN#'`, `'ics#'`, and `'icm#'` resources.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAll4BitData`
Specify to include `'icl4'`, `'ics4'`, and `'icm4'` resources.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAll8BitData`
Specify to include `'icl8'`, `'ics8'`, and `'icm8'` resources.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAll32BitData`
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

`kSelectorAllAvailableData`
Specify to include all resources of given ID.
Available in Mac OS X v10.0 and later.
Declared in `Icons.h`.

Discussion
The functions `GetIconSuite` (page 1257) and `ForEachIconDo` (page 1244) use these constants in the `selector` parameter to specify which members of the family to include in the icon suite.

Catalog Information Bitmask

Defines a minimal bitmask for use with the `GetIconRefFromFileInfo` function.

```
enum {
    kIconServicesCatalogInfoMask =
            (kFSCatInfoNodeID | kFSCatInfoParentDirID | kFSCatInfoVolume
            | kFSCatInfoNodeFlags | kFSCatInfoFinderInfo |
            kFSCatInfoFinderXInfo | kFSCatInfoUserAccess)
};
```

Constants

`kIconServicesCatalogInfoMask`

Use this mask with the File Manager function `FSGetCatalogInfo` before calling `GetIconRefFromFileInfo`.

Available in Mac OS X v10.1 and later.

Declared in `IconsCore.h`.

System Icon Constant

Defines a creator type for all system–defined icons.

```
enum {
    kSystemIconsCreator = 'macs'
};
```

Discussion

You can use the `kSystemIconsCreator` constant to obtain System icons that are not associated with a file, such as the help icon.

Icon Services Usage Flag

```
typedef UInt32 IconServicesUsageFlags;
enum {
    kIconServicesNormalUsageFlag = 0
};
```

Alert Icon Constants

Specify standard alert icons.

```
enum {
    kAlertNoteIcon = 'note',
    kAlertCautionIcon = 'caut',
    kAlertStopIcon = 'stop'
};
```

Discussion

Icon Services defines constants for a number of standard alert icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Filesharing Privilege Icon Constants

Identify standard filesharing privilege icons.

```
enum {
    kSharingPrivsNotApplicableIcon = 'shna',
    kSharingPrivsReadOnlyIcon = 'shro',
    kSharingPrivsReadWriteIcon = 'shrw',
    kSharingPrivsUnknownIcon = 'shuk',
    kSharingPrivsWritableIcon = 'writ'
};
```

Discussion
Icon Services defines constants for a number of standard filesharing privilege icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Folder Icon Constants

Identify standard folder icons.

```
enum {
    kGenericFolderIcon = 'fldr',
    kDropFolderIcon = 'dbox',
    kMountedFolderIcon = 'mntd',
    kOpenFolderIcon = 'ofld',
    kOwnedFolderIcon = 'ownd',
    kPrivateFolderIcon = 'prvf',
    kSharedFolderIcon = 'shfl'
};
```

Discussion
Icon Services defines constants for a number of standard folder icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Internet Icon Constants

Identify standard Internet icons.

```
enum {
    kInternetLocationHTTPIcon = 'ilht',
    kInternetLocationFTPIcon = 'ilft',
    kInternetLocationAppleShareIcon = 'ilaf',
    kInternetLocationAppleTalkZoneIcon = 'ilat',
    kInternetLocationFileIcon = 'ilfi',
    kInternetLocationMailIcon = 'ilma',
    kInternetLocationNewsIcon = 'ilnw',
    kInternetLocationNSLNeighborhoodIcon = 'ilns',
    kInternetLocationGenericIcon = 'ilge'
};
```

Discussion
Icon Services defines constants for a number of standard Internet icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Toolbar Icons

Identify standard toolbar icons.

```
enum {
    kToolbarCustomizeIcon = 'tcus',
    kToolbarDeleteIcon = 'tdel',
    kToolbarFavoritesIcon = 'tfav',
    kToolbarHomeIcon = 'thom'
};
```

Miscellaneous Icon Constants

Identify miscellaneous icons.

```
enum {
    kAppleLogoIcon = 'capl',
    kAppleMenuIcon = 'sapl',
    kBackwardArrowIcon = 'baro',
    kFavoriteItemsIcon = 'favr',
    kForwardArrowIcon = 'faro',
    kGridIcon = 'grid',
    kHelpIcon = 'help',
    kKeepArrangedIcon = 'arng',
    kLockedIcon = 'lock',
    kNoFilesIcon = 'nfil',
    kNoFolderIcon = 'nfld',
    kNoWriteIcon = 'nwrt',
    kProtectedApplicationFolderIcon = 'papp',
    kProtectedSystemFolderIcon = 'psys',
    kRecentItemsIcon = 'rcnt',
    kShortcutIcon = 'shrt',
    kSortAscendingIcon = 'asnd',
    kSortDescendingIcon = 'dsnd',
    kUnlockedIcon = 'ulck',
    kConnectToIcon = 'cnct',
    kGenericWindowIcon = 'gwin',
    kQuestionMarkIcon = 'ques',
    kDeleteAliasIcon = 'dali',
    kEjectMediaIcon = 'ejec',
    kBurningIcon = 'burn',
    kRightContainerArrowIcon = 'rcar'
};
```

Discussion
Icon Services defines constants for a number of miscellaneous icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Networking Icon Constants

Identify standard networking icons.

```
enum {
    kAppleTalkIcon = 'atlk',
    kAppleTalkZoneIcon = 'atzn',
    kAFPServerIcon = 'afps',
    kFTPServerIcon = 'ftps',
    kHTTPServerIcon = 'https',
    kGenericNetworkIcon = 'gnet',
    kIPFileServerIcon = 'isrv'
};
```

Discussion
Icon Services defines constants for a number of standard networking icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Special Folder Icon Constants

Identify special folder icons.

```
enum {
    kAppearanceFolderIcon = 'appr',
    kAppleExtrasFolderIcon = 'aexf',
    kAppleMenuFolderIcon = 'amnu',
    kApplicationsFolderIcon = 'apps',
    kApplicationSupportFolderIcon = 'asup',
    kAssistantsFolderIcon = 'astf',
    kColorSyncFolderIcon = 'prof',
    kContextualMenuItemsFolderIcon = 'cmnu',
    kControlPanelDisabledFolderIcon = 'ctrD',
    kControlPanelFolderIcon = 'ctrl',
    kControlStripModulesFolderIcon = 'sdvf',
    kDocumentsFolderIcon = 'docs',
    kExtensionsDisabledFolderIcon = 'extD',
    kExtensionsFolderIcon = 'extn',
    kFavoritesFolderIcon = 'favs',
    kFontsFolderIcon = 'font',
    kHelpFolderIcon = 'fhlp',
    kInternetFolderIcon = 'intf',
    kInternetPlugInFolderIcon = 'fnet',
    kInternetSearchSitesFolderIcon = 'issf',
    kLocalesFolderIcon = 'floc',
    kMacOSReadMeFolderIcon = 'morf',
    kPublicFolderIcon = 'pubf',
    kPreferencesFolderIcon = 'prff',
    kPrinterDescriptionFolderIcon = 'ppdf',
    kPrinterDriverFolderIcon = 'fprd',
    kPrintMonitorFolderIcon = 'prnt',
    kRecentApplicationsFolderIcon = 'rapp',
    kRecentDocumentsFolderIcon = 'rdoc',
    kRecentServersFolderIcon = 'rsrv',
    kScriptingAdditionsFolderIcon = 'fscr',
    kSharedLibrariesFolderIcon = 'flib',
    kScriptsFolderIcon = 'scrf',
    kShutdownItemsDisabledFolderIcon = 'shdD',
    kShutdownItemsFolderIcon = 'shdf',
    kSpeakableItemsFolder = 'spki',
    kStartupItemsDisabledFolderIcon = 'strD',
    kStartupItemsFolderIcon = 'strt',
    kSystemExtensionDisabledFolderIcon = 'macD',
    kSystemFolderIcon = 'macs',
    kTextEncodingsFolderIcon = 'ftex',
    kUsersFolderIcon = 'usrf',
    kUtilitiesFolderIcon = 'utif',
    kVoicesFolderIcon = 'fvoc'
};
```

Discussion
Icon Services defines constants for a number of special folder icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Standard Finder Icon Constants

Identify standard Finder icons.

```
enum {
    kClipboardIcon = 'CLIP',
    kClippingUnknownTypeIcon = 'clpu',
    kClippingPictureTypeIcon = 'clpp',
    kClippingTextTypeIcon = 'clpt',
    kClippingSoundTypeIcon = 'clps',
    kDesktopIcon = 'desk',
    kFinderIcon = 'FNDR',
    kFontSuitcaseIcon = 'FFIL',
    kFullTrashIcon = 'ftrh',
    kGenericApplicationIcon = 'APPL',
    kGenericCDROMIcon = 'cddr',
    kGenericControlPanelIcon = 'APPC',
    kGenericControlStripModuleIcon = 'sdev',
    kGenericComponentIcon = 'thng',
    kGenericDeskAccessoryIcon = 'APPD',
    kGenericDocumentIcon = 'docu',
    kGenericEditionFileIcon = 'edtf',
    kGenericExtensionIcon = 'INIT',
    kGenericFileServerIcon = 'srvr',
    kGenericFontIcon = 'ffil',
    kGenericFontScalerIcon = 'sclr',
    kGenericFloppyIcon = 'flpy',
    kGenericHardDiskIcon = 'hdsk',
    kGenericIDiskIcon = 'idsk',
    kGenericRemovableMediaIcon = 'rmov',
    kGenericMoverObjectIcon = 'movr',
    kGenericPCCardIcon = 'pcmc',
    kGenericPreferencesIcon = 'pref',
    kGenericQueryDocumentIcon = 'qery',
    kGenericRAMDiskIcon = 'ramd',
    kGenericSharedLibaryIcon = 'shlb',
    kGenericStationeryIcon = 'sdoc',
    kGenericSuitcaseIcon = 'suit',
    kGenericURLIcon = 'gurl',
    kGenericWORMIcon = 'worm',
    kInternationalResourcesIcon = 'ifil',
    kKeyboardLayoutIcon = 'kfil',
    kSoundFileIcon = 'sfil',
    kSystemSuitcaseIcon = 'zsys',
    kTrashIcon = 'trsh',
    kTrueTypeFontIcon = 'tfil',
    kTrueTypeFlatFontIcon = 'sfnt',
    kTrueTypeMultiFlatFontIcon = 'ttcf',
    kUserIDiskIcon = 'udsk',
    kUnknownFSObjectIcon = 'unfs',
    kInternationResourcesIcon = kInternationalResourcesIcon
};
```

Discussion
Icon Services defines constants for a number of standard Finder icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Standard Icon Badge Constants

Identify standard badges.

```
enum {
    kAppleScriptBadgeIcon = 'scrp',
    kLockedBadgeIcon = 'lbdg',
    kMountedBadgeIcon = 'mbdg',
    kSharedBadgeIcon = 'sbdg',
    kAliasBadgeIcon = 'abdg',
    kAlertCautionBadgeIcon = 'cbdg'
};
```

Discussion
Icon Services defines constants for a number of standard badges. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

Users and Groups Icon Constants

Identify icons used in the Users and Groups control panel.

```
enum {
    kUserFolderIcon = 'ufld',
    kWorkgroupFolderIcon = 'wfld',
    kGuestUserIcon = 'gusr',
    kUserIcon = 'user',
    kOwnerIcon = 'susr',
    kGroupIcon = 'grup'
};
```

Discussion
Icon Services defines constants for a number of icons used in the Users and Groups control panel. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 1249), for example.

genericDocumentIconResource

Use the constants listed in "Standard Icon Resources" instead.

```
enum {
    genericDocumentIconResource = kGenericDocumentIconResource,
    genericStationeryIconResource = kGenericStationeryIconResource,
    genericEditionFileIconResource = kGenericEditionFileIconResource,
    genericApplicationIconResource = kGenericApplicationIconResource,
    genericDeskAccessoryIconResource = kGenericDeskAccessoryIconResource,
    genericFolderIconResource = kGenericFolderIconResource,
    privateFolderIconResource = kPrivateFolderIconResource,
    floppyIconResource = kFloppyIconResource,
    trashIconResource = kTrashIconResource,
    genericRAMDiskIconResource = kGenericRAMDiskIconResource,
    genericCDROMIconResource = kGenericCDROMIconResource,
    desktopIconResource = kDesktopIconResource,
    openFolderIconResource = kOpenFolderIconResource,
    genericHardDiskIconResource = kGenericHardDiskIconResource,
    genericFileServerIconResource = kGenericFileServerIconResource,
    genericSuitcaseIconResource = kGenericSuitcaseIconResource,
    genericMoverObjectIconResource = kGenericMoverObjectIconResource,
    genericPreferencesIconResource = kGenericPreferencesIconResource,
    genericQueryDocumentIconResource = kGenericQueryDocumentIconResource,
    genericExtensionIconResource = kGenericExtensionIconResource,
    systemFolderIconResource = kSystemFolderIconResource,
    appleMenuFolderIconResource = kAppleMenuFolderIconResource
};
```

Standard Icon Resources

Identify standard icon resources.

```
/*Icons for which both icon suites and 'SICN' resources exist*/
enum {
    kGenericDocumentIconResource = -4000,
    kGenericStationeryIconResource = -3985,
    kGenericEditionFileIconResource = -3989,
    kGenericApplicationIconResource = -3996,
    kGenericDeskAccessoryIconResource = -3991,
    kGenericFolderIconResource = -3999,
    kPrivateFolderIconResource = -3994,
    kFloppyIconResource = -3998,
    kTrashIconResource = -3993,
    kGenericRAMDiskIconResource = -3988,
    kGenericCDROMIconResource = -3987
};
/* Icons for which only 'SICN' resources exist*/
enum {
    kDesktopIconResource = -3992,
    kOpenFolderIconResource = -3997,
    kGenericHardDiskIconResource = -3995,
    kGenericFileServerIconResource = -3972,
    kGenericSuitcaseIconResource = -3970,
    kGenericMoverObjectIconResource = -3969
};
/*Icons for which only icon suites exist*/
enum {
    kGenericPreferencesIconResource = -3971,
    kGenericQueryDocumentIconResource = -16506,
    kGenericExtensionIconResource = -16415,
    kSystemFolderIconResource = -3983,
    kHelpIconResource = -20271,
    kAppleMenuFolderIconResource = -3982
};
enum {
    kStartupFolderIconResource = -3981,
    kOwnedFolderIconResource = -3980,
    kDropFolderIconResource = -3979,
    kSharedFolderIconResource = -3978,
    kMountedFolderIconResource = -3977,
    kControlPanelFolderIconResource = -3976,
    kPrintMonitorFolderIconResource = -3975,
    kPreferencesFolderIconResource = -3974,
    kExtensionsFolderIconResource = -3973,
    kFontsFolderIconResource = -3968,
    kFullTrashIconResource = -3984
};
```

startupFolderIconResource

Use the constants described in "Standard Icon Resources" instead.

```
enum {
    startupFolderIconResource = kStartupFolderIconResource,
    ownedFolderIconResource = kOwnedFolderIconResource,
    dropFolderIconResource = kDropFolderIconResource,
    sharedFolderIconResource = kSharedFolderIconResource,
    mountedFolderIconResource = kMountedFolderIconResource,
    controlPanelFolderIconResource = kControlPanelFolderIconResource,
    printMonitorFolderIconResource = kPrintMonitorFolderIconResource,
    preferencesFolderIconResource = kPreferencesFolderIconResource,
    extensionsFolderIconResource = kExtensionsFolderIconResource,
    fontsFolderIconResource = kFontsFolderIconResource,
    fullTrashIconResource = kFullTrashIconResource
};
```

atNone

Use the constants described in "Icon Alignment Constants" instead.

```
enum {
    atNone = kAlignNone,
    atVerticalCenter = kAlignVerticalCenter,
    atTop = kAlignTop,
    atBottom = kAlignBottom,
    atHorizontalCenter = kAlignHorizontalCenter,
    atAbsoluteCenter = kAlignAbsoluteCenter,
    atCenterTop = kAlignCenterTop,
    atCenterBottom = kAlignCenterBottom,
    atLeft = kAlignLeft,
    atCenterLeft = kAlignCenterLeft,
    atTopLeft = kAlignTopLeft,
    atBottomLeft = kAlignBottomLeft,
    atRight = kAlignRight,
    atCenterRight = kAlignCenterRight,
    atTopRight = kAlignTopRight,
    atBottomRight = kAlignBottomRight
};
```

svLarge1Bit

Use the constants described in "Icon Selector Constants" instead.

```
enum {
    svLarge1Bit = kSelectorLarge1Bit,
    svLarge4Bit = kSelectorLarge4Bit,
    svLarge8Bit = kSelectorLarge8Bit,
    svSmall1Bit = kSelectorSmall1Bit,
    svSmall4Bit = kSelectorSmall4Bit,
    svSmall8Bit = kSelectorSmall8Bit,
    svMini1Bit = kSelectorMini1Bit,
    svMini4Bit = kSelectorMini4Bit,
    svMini8Bit = kSelectorMini8Bit,
    svAllLargeData = kSelectorAllLargeData,
    svAllSmallData = kSelectorAllSmallData,
    svAllMiniData = kSelectorAllMiniData,
    svAll1BitData = kSelectorAll1BitData,
    svAll4BitData = kSelectorAll4BitData,
    svAll8BitData = kSelectorAll8BitData,
    svAllAvailableData = kSelectorAllAvailableData
};
```

ttNone

Use the constants described in "Icon Transformation Constants" instead.

```
enum {
    ttNone = kTransformNone,
    ttDisabled = kTransformDisabled,
    ttOffline = kTransformOffline,
    ttOpen = kTransformOpen,
    ttLabel1 = kTransformLabel1,
    ttLabel2 = kTransformLabel2,
    ttLabel3 = kTransformLabel3,
    ttLabel4 = kTransformLabel4,
    ttLabel5 = kTransformLabel5,
    ttLabel6 = kTransformLabel6,
    ttLabel7 = kTransformLabel7,
    ttSelected = kTransformSelected,
    ttSelectedDisabled = kTransformSelectedDisabled,
    ttSelectedOffline = kTransformSelectedOffline,
    ttSelectedOpen = kTransformSelectedOpen
};
```

Result Codes

The table below shows the most common result codes returned by Icon Services and Utilities.

Result Code	Value	Description
`noMaskFoundErr`	-1000	Available in Mac OS X v10.0 and later.
`invalidIconRefErr`	-2580	The `IconRef` is not valid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`noSuchIconErr`	-2581	The requested icon could not be found. Available in Mac OS X v10.0 and later.
`noIconDataAvailableErr`	-2582	The necessary icon data is not available. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Icon Services selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Language Analysis Manager Reference

Framework: ApplicationServices/ApplicationServices.h
Declared in LanguageAnalysis.h

Overview

The Language Analysis Manager application programming interface (API) is a shared library designed to analyze morphemes in text. It is a general-purpose API that does not rely on languages, algorithms of morpheme analysis, or their applications. Language Analysis Manager is not a framework for creating International-aware applications. To make your applications work correctly with various languages, you can use APIs such as Script Manager and Text Utilities.

The Language Analysis Manager (LAM) provides your application with morphological analysis capability, and is designed to work with a language analysis engine. Using the Language Analysis Manager, your application can manage an analysis engine and create environments and contexts in which morpheme analysis can occur. This version of the Language Analysis Manager works only with a Japanese analysis engine.

Functions by Task

Getting The Library Version

`LALibraryVersion` (page 1335) Deprecated in Mac OS X v10.5
Returns the version of the Language Analysis Manager installed.

Handling Environments

`LACreateCustomEnvironment` (page 1332) Deprecated in Mac OS X v10.5
Creates a new environment with the specified name.

`LADeleteCustomEnvironment` (page 1332) Deprecated in Mac OS X v10.5
Disposes of a reference to a custom language analysis environment.

`LAGetEnvironmentList` (page 1333) Deprecated in Mac OS X v10.5
Obtains a list of the available language analysis environments.

`LAGetEnvironmentName` (page 1334) Deprecated in Mac OS X v10.5
Obtains the name of an environment.

`LAGetEnvironmentRef` (page 1334) Deprecated in Mac OS X v10.5
Obtains the language analysis environment reference associated with an environment name

Opening and Closing Contexts

`LACloseAnalysisContext` (page 1329) Deprecated in Mac OS X v10.5
Closes the specified language analysis context.

`LAOpenAnalysisContext` (page 1338) Deprecated in Mac OS X v10.5
Creates a language analysis context from a specified language analysis environment.

Managing Dictionaries

`LAAddNewWord` (page 1328) Deprecated in Mac OS X v10.5
Adds a new word to a dictionary.

`LACloseDictionary` (page 1330) Deprecated in Mac OS X v10.5
Closes a dictionary in the specified environment.

`LAListAvailableDictionaries` (page 1336) Deprecated in Mac OS X v10.5
Obtains the number of dictionaries available in a specified environment.

`LAOpenDictionary` (page 1339) Deprecated in Mac OS X v10.5
Opens a dictionary for the specified environment.

Analyzing Text

`LAContinuousMorphemeAnalysis` (page 1330) Deprecated in Mac OS X v10.5
Performs a continuous morphological analysis of Unicode text.

`LAGetMorphemes` (page 1335) Deprecated in Mac OS X v10.5
Reads the results of a continuous morpheme analysis.

`LAMorphemeAnalysis` (page 1337) Deprecated in Mac OS X v10.5
Performs a morphological analysis of the specified Unicode text.

`LAResetAnalysis` (page 1339) Deprecated in Mac OS X v10.5
Clears the internal status of the analysis context.

`LAShiftMorphemes` (page 1340) Deprecated in Mac OS X v10.5
Shifts the read out of continuous morpheme analysis.

`LATextToMorphemes` (page 1341) Deprecated in Mac OS X v10.5
Performs a morphological analysis of the specified text.

Functions

LAAddNewWord

Adds a new word to a dictionary. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAAddNewWord (
   LAEnvironmentRef environ,
   const FSSpec *dictionary,
   const AEDesc *dataList
);
```

Parameters

environ

A reference to the language analysis environment for the dictionary you want to modify.

dictionary

The file specification for the dictionary you want to modify.

dataList

A pointer to an `AEDesc` data structure that specifies the word you want to add to the dictionary. See the Apple Event Manager documentation for more information on Apple Event descriptor records.

Return Value
A result code. See "Result Codes" (page 1354).

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LACloseAnalysisContext

Closes the specified language analysis context. (Deprecated in Mac OS X v10.5.)

```
OSStatus LACloseAnalysisContext (
   LAContextRef context
);
```

Parameters

context

A reference to the language analysis context you want to close.

Return Value
A result code. See "Result Codes" (page 1354).

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LACloseDictionary

Closes a dictionary in the specified environment. (Deprecated in Mac OS X v10.5.)

```
OSStatus LACloseDictionary (
   LAEnvironmentRef environ,
   const FSSpec *dictionary
);
```

Parameters

environ

A reference to the language analysis environment for which you want to close a dictionary.

dictionary

The file specification for the dictionary you want to close.

Return Value

A result code. See "Result Codes" (page 1354).

Availability

Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`LanguageAnalysis.h`

LAContinuousMorphemeAnalysis

Performs a continuous morphological analysis of Unicode text. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAContinuousMorphemeAnalysis (
   LAContextRef context,
   ConstUniCharArrayPtr text,
   UniCharCount textLength,
   Boolean incrementalText,
   LAMorphemePath *leadingPath,
   LAMorphemePath *trailingPath,
   Boolean *modified
);
```

Parameters

context

A reference to the language analysis context whose text you want to analyze. You can obtain a language analysis context by calling the function `LAOpenAnalysisContext`.

text

A pointer to the Unicode text string you want to analyze.

textLength

The length of the Unicode text string specified in the `text` parameter. This value must specify the number of `UniChar` (double-byte) values in the string.

incrementalText

A Boolean value that indicates the method for passing text. Pass `false` to specify you want the text to be analyzed as a whole and the analysis started. Pass `true` if the text is a continuation of the text currently held by the context, and should be added to the context before undergoing analysis.

leadingPath

A pointer to the morpheme path that specifies the results of analyzing the text just previous to the string specified by the `text` parameter. The Langauage Analysis Manager uses this string to restrict the analyis. For example, if the previous section ends with a noun, the text that follows can begin with a verb. If no valid leading path is available you can pass `NULL` or a "Leading and Trailing Constants" (page 1350)—`kLAFreeEdge` or `kLADefaultEdge`. Pass `kLAFreeEdge` if it is possible for an optional morpheme to come at the start or the end of analysis. Pass `kLADefaultEdge` if you want the analysis is carried out so that the start/end of analysis becomes the start of the sentence/end of sentence or the start of the segment/end of segment. Definitions for start of sentence/end of sentence and start of segment/end of segment depend on the engine.

trailingPath

A pointer to the morpheme path that specifies the results of analyzing the text that follows the string specified by the `text` parameter. When performing a continuous analysis, you must pass the constant `kLAIncompleteEdge` to indicate that the string is not complete. Note that the function `LAGetMorphemes` only returns the results it has completed analyzing, not the analysis of the complete source text. If you want to obtain all of the analysis results up to a point, (if a user expressly indicates a conversion with the space bar in a kana-kanji conversion program, and so forth) then you can pass a value other than `kLAIncompleteEdge`. Then, when you call the function `LAGetMorphemes` you obtain analysis results for that portion of the string that has been analyzed to that point.

modified

On output, `true` if the internal state of the context is changed (new analyzed morphemes are generated); otherwise `false`. When `true` is returned, you should call the function `LAGetMorphemes` and update the display. If modified is specified as `NULL`, values are not returned.

Return Value
A result code. See "Result Codes" (page 1354).

Discussion
The function `LAContinuousMorphemeAnalysis` does not return analysis results, but holds them internally. You can obtain the results by calling the functions `LAGetMorphemes` or `LAShiftMorphemes`. In contrast to the function `LAMorphemeAnalysis` you cannot obtain multiple paths for an analysis done using the function `LAContinuousMorphemeAnalysis`.

You can obtain the same results as calling the function `LAResetAnalysis` by calling `LAContinuousMorphemeAnalysis` with the `text` parameter set to "" and the `incrementalText` parameter set to `false`.

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LACreateCustomEnvironment

Creates a new environment with the specified name. (Deprecated in Mac OS X v10.5.)

```
OSStatus LACreateCustomEnvironment (
   LAEnvironmentRef baseEnvironment,
   ConstStr63Param newEnvironmentName,
   Boolean persistent,
   LAEnvironmentRef *newEnvironment
);
```

Parameters

baseEnvironment

A reference to the language analysis environment that you want to use as the base environment.

newEnvironmentName

The name for the newly-created environment. This name must be unique. If an environment with the same name already exists, the function returns the result code `laEnvironmentExistErr`.

persistent

A Boolean value that specifies whether the environment should be persistent (`true`) or not (`false`). If you pass `true`, the newly-created environment is saved to disk, and it can be referred to at any time subsequently by using the name. If you pass `false`, the newly-created environment can only be used during that session. Additionally, environments created with persistent set to `false` are not returned in the list provided by the function `LAGetEnvironmentList`, so these environments can be used only as private environments. If you create a private environment, you must call the function `LADeleteCustomEnviroment` to dispose of it before you terminate your application.

newEnvironemnt

On output, a reference to the newly-created language analysis environment.

Return Value

A result code. See "Result Codes" (page 1354).

Discussion

If you open or close dictionaries for custom environments, it is possible to create independent environments without interfering with existing environments.

Availability

Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`LanguageAnalysis.h`

LADeleteCustomEnvironment

Disposes of a reference to a custom language analysis environment. (Deprecated in Mac OS X v10.5.)

```
OSStatus LADeleteCustomEnvironment (
   LAEnvironmentRef environment
);
```

Parameters

environment

A reference to the language analysis environment you want to dispose of.

Return Value
A result code. See "Result Codes" (page 1354).

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAGetEnvironmentList

Obtains a list of the available language analysis environments. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAGetEnvironmentList (
   UInt32 maxCount,
   UInt32 *actualCount,
   LAEnvironmentRef environmentList[]
);
```

Parameters

maxCount

The maximum number of environments provided by the system. To determine this value, see the Discussion.

actualCount

On output, the actual number of environments.

environmentList

On output, a list of the available environments. You must allocate a buffer of the appropriate size. If you are uncertain of how much memory to allocate for this array, see the Discussion.

Return Value
A result code. See "Result Codes" (page 1354).

Discussion
Typically, you use the function `LAGetEnvironmentList` by calling it twice, as follows:

1. Pass `0` for the `maxCount` parameter and `NULL` for the `environmentList` parameter.
2. Allocate enough space for an array of the size specified by `actualCount`, then call the function `LAGetEnvironmentList` again. This time, provide a count of the actual number of environments as the `maxCount` parameter, and a pointer to a buffer of the correct size for the `environmentList` parameter. On output, the pointer points to an array of the available environments.

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAGetEnvironmentName

Obtains the name of an environment. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAGetEnvironmentName (
   LAEnvironmentRef environment,
   Str63 environmentName
);
```

Parameters

environment
A reference to the language analysis environment whose name you want to obtain.

environmentName
On return, the environment name.

Return Value
A result code. See "Result Codes" (page 1354).

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAGetEnvironmentRef

Obtains the language analysis environment reference associated with an environment name (Deprecated in Mac OS X v10.5.)

```
OSStatus LAGetEnvironmentRef (
   ConstStr63Param targetEnvironmentName,
   LAEnvironmentRef *environment
);
```

Parameters

targetEnvironmentName
The environment name whose language analysis environment reference you want to obtain.

environment
On output, a reference to the language analysis environment associated with the environment name.

Return Value
A result code. See "Result Codes" (page 1354).

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAGetMorphemes

Reads the results of a continuous morpheme analysis. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAGetMorphemes (
   LAContextRef context,
   LAMorphemePath *result
);
```

Parameters

context

A reference to the language analysis context whose result you want to obtain. You can obtain a language analysis context by calling the function `LAOpenAnalysisContext`.

result

On output, points to the morpheme bundle that contains the results of the analysis. You are responsible for disposing of this structure by calling the Apple Event Manager function `AEDisposeDesc`.

Return Value
A result code. See "Result Codes" (page 1354).

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LALibraryVersion

Returns the version of the Language Analysis Manager installed. (Deprecated in Mac OS X v10.5.)

```
UInt32 LALibraryVersion (
   void
);
```

Return Value
Returns the version of Language Analysis manager that is installed.

Discussion
The function `LALibraryVersion` returns the version of the Language Analysis Manager installed in the same format as `'vers'` resource. That is to say, the version number is returned in BCD (Binary-Coded Decimal) format to higher-place words, while release stage information is returned to lower-place words. For example, version 1.1.1 final release library returns 0x01118000.

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAListAvailableDictionaries

Obtains the number of dictionaries available in a specified environment. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAListAvailableDictionaries (
   LAEnvironmentRef environ,
   ItemCount maxCount,
   ItemCount *actualCount,
   FSSpec dictionaryList[],
   Boolean opened[]
);
```

Parameters

environ
A reference to the language analysis environment for which you want to obtain a list of available dictionaries.

maxCount
The maximum number of available dictionaries. To determine this value, see the Discussion.

actualCount
On output, the actual number of available dictionaries.

dictionaryList
On output, points to a list of available dictionaries. You must allocate a buffer of the appropriate size. If you are uncertain of how much memory to allocate for this array, see the Discussion.

opened
On output, points to a list of Boolean values that specify whether the available dictionaries are open. This array is parallel to the `dictionaryList` array. A dictionary file whose associated value is `true` is open and `false` if it is not open. You must allocate a buffer of the appropriate size. If you are uncertain of how much memory to allocate for this array, see the Discussion.

Return Value
A result code. See "Result Codes" (page 1354).

Discussion
Typically, you use the function `LAListAvailableDictionaries` by calling it twice, as follows:

1. Pass `0` for the `maxCount` parameter, `NULL` for the `dictionaryList` parameter, and `NULL` for the `opened` parameter.

2. Allocate enough space for arrays of the size specified by `actualCount`, then call the function `LAListAvailableDictionaries` again. This time, provide a count of the actual number of dictionaries as the `maxCount` parameter, and a pointer to buffers of the correct size for the `dictionaryList` and `opened` parameters. On output, `dictionaryList` points to an array of the available dictionaries and `opened` points to an array that specifies whether each dictionary is opened or closed.

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAMorphemeAnalysis

Performs a morphological analysis of the specified Unicode text. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAMorphemeAnalysis (
   LAContextRef context,
   ConstUniCharArrayPtr text,
   UniCharCount textLength,
   LAMorphemePath *leadingPath,
   LAMorphemePath *trailingPath,
   ItemCount pathCount,
   LAMorphemeBundle *result
);
```

Parameters

context
A reference to the language analysis context whose text you want to analyze. You can obtain a language analysis context by calling the function `LAOpenAnalysisContext`.

text
A pointer to the Unicode text string you want to analyze.

textLength
The length of the Unicode text string specified in the `text` parameter. This value must specify the number of `UniChar` (double-byte) values in the string.

leadingPath
A pointer to the morpheme path that specifies the results of analyzing the text just previous to the string specified by the `text` parameter. The Language Analysis Manager uses this string to restrict the analysis. For example, if the previous section ends with a noun, the text that follows can begin with a verb. If no valid leading path is available you can pass `NULL` or a "Leading and Trailing Constants" (page 1350)—`kLAFreeEdge` or `kLADefaultEdge`. Pass `kLAFreeEdge` if it is possible for an optional morpheme to come at the start or the end of analysis. Pass `kLADefaultEdge` if you want the analysis is carried out so that the start/end of analysis becomes the start of the sentence/end of sentence or the start of the segment/end of segment. Definitions for start of sentence/end of sentence and start of segment/end of segment depend on the engine.

trailingPath

A pointer to the morpheme path that specifies the results of analyzing the text that follows the string specified by the `text` parameter. The Language Analysis Manager uses this string to restrict the analysis. For example, if the following section begins with a verb, the text that precedes it can begin with a noun. If no valid trailing path is available you can pass `NULL` or a "Leading and Trailing Constants" (page 1350)—`kLAFreeEdge` or `kLADefaultEdge`. Pass `kLAFreeEdge` if it is possible for an optional morpheme to come at the start or the end of analysis. Pass `kLADefaultEdge` if you want the analysis is carried out so that the start/end of analysis becomes the start of the sentence/end of sentence or the start of the segment/end of segment. Definitions for start of sentence/end of sentence and start of segment/end of segment depend on the engine.

pathCount

On output, specifies the maximum rank of the returned path.

result

On output, points to the morpheme bundle that contains the results of the analysis. You are responsible for disposing of this structure by calling the Apple Event Manager function `AEDisposeDesc`.

Return Value
A result code. See "Result Codes" (page 1354).

Discussion
If you have previously called the function `LAContinuousMorphemeAnalysis`, and you the call the function `LAMorphemeAnalysis`, the internal state maintained by the function `LAContinuousMorphemeAnalysis` is disposed of. Then, if you call the functions `LAGetMorphemes` and `LAShiftMorphemes` the result code `laNoMoreMorphemeErr` is returned .

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAOpenAnalysisContext

Creates a language analysis context from a specified language analysis environment. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAOpenAnalysisContext (
   LAEnvironmentRef environ,
   LAContextRef *context
);
```

Parameters

environ

A reference to the language analysis environment for which you want to open a context.

context

On output, a language analysis context derived from the specified language analysis environment.

Return Value
A result code. See "Result Codes" (page 1354).

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAOpenDictionary

Opens a dictionary for the specified environment. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAOpenDictionary (
   LAEnvironmentRef environ,
   const FSSpec *dictionary
);
```

Parameters

environ
A reference to the language analysis environment for which you want to open the dictionary.

dictionary
The file specification for the dictionary you want to open.

Return Value
A result code. See "Result Codes" (page 1354).

Discussion
The environment makes an appropriate assessment of type of dictionary, user dictionary, option dictionary and so forth, before carrying out necessary operations.

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAResetAnalysis

Clears the internal status of the analysis context. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAResetAnalysis (
   LAContextRef context
);
```

Parameters

context
A reference to the language analysis context whose analysis you want to reset. You can obtain a language analysis context by calling the function `LAOpenAnalysisContext`.

Return Value
A result code. See "Result Codes" (page 1354).

Discussion
Clear the internal status of analysis context. This is accessed before the continuous analysis by the next LAContinuousMorphemeAnalysis. Accessing LAGetMorphemes and LAShiftMorphemes immediately after this call will fail.

The same result will be achieved even if LAContinuousMorphemeAnalysis is accessed as text = "", incrementalText = false

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAShiftMorphemes

Shifts the read out of continuous morpheme analysis. (Deprecated in Mac OS X v10.5.)

```
OSStatus LAShiftMorphemes (
   LAContextRef context,
   ItemCount morphemeCount,
   LAMorphemePath *path,
   UniCharCount *shiftedLength
);
```

Parameters

context
A reference to the language analysis context whose read-out you want to shift. You can obtain a language analysis context by calling the function `LAOpenAnalysisContext`.

morphemeCount
The number of morphemes to be shifted. If you pass `kAllMorphemes`, all morphemes which are analized are returned.

path
If you pass `typeNull` a new path is created. If you pass a valid path, the morpheme read out at the end of the path is added. This is handy when this path is to be used as the leading edge the next time `LAContinuousMorphemeAnalysis` is accessed. In both cases, when you are done using the path, you must dispose of it by calling the Apple Event Manager function `AEDispose`.

shiftedLength
A pointer to the input character string length (in `UniChars`) corresponding to the morpheme read out. If you pass `NULL`, it is not returned.

Return Value
A result code. See "Result Codes" (page 1354).

Discussion
When you call the function `LAShiftMorphemes`, the results of the analysis performed by the function `LAContinuousMorphemeAnalysis` are returned from the start to the number of morpheme readout paths specified in the `morphemeCount` parameter . Morphemes which have been read out are deleted from analysis context. For example, if "AABBCC" represents the internal status, after you fetch the morpheme "AA" by calling `LAShiftMorphemes`, the internal status becomes "BBCC".

The results you obtain by calling the funciton `LAShiftMorphemes`, are impacted by the `trailingEdge` parameter of the function `LAContinuousMorphemeAnalysis`. If the value of the `trailingEdge` parameter is `kLAIncompleteEdge`, the function `LAShiftMorphemes` might not return all morphemes and non-converted sections. The analysis engine only returns morphemes with a high degree of certainty. For example, those morphemes which are likely to change if text is added, and `laNoMoreMorphemeErr` is returned in subsequent accesses. If something other than `kLAIncompleteEdge` is passed as the `trailingEdge` parameter, it is possible to fetch morphemes up to the final morpheme. After all morphemes are fetched, the result code `laNoMoreMorphemeErr` is returned to indicate that nothing remains. This is the same as the status returned after calling the function `LAResetAnalysis`.

You can carry out a continuous analysis using the functions `LAContinuousMorphemeAnalysis` and `LAShiftMorphemes` in two ways. The first method leaves as much text as possible within the analysis engine. That is, continue to provide text to the fucntoin `LAContinuousMorphemeAnalysis` until you encounter the result code `laTextOverflowErr` , and call the function `LAShiftMorphemes` once when the error is returned. The second method leaves as little text as possible within the analysis search engine. That is, continue to provide text to the function `LAContinuousMorphemeAnalysis` until `true` is returned to the `modified` parameter. When `true` is returned, call the function `LAShiftMorphemes` with the `morphemeCount` parameter set to `kAllMorphemes`.

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LATextToMorphemes

Performs a morphological analysis of the specified text. (Deprecated in Mac OS X v10.5.)

```
OSStatus LATextToMorphemes (
   LAContextRef context,
   TextEncoding preferedEncoding,
   ByteCount textLength,
   ConstLogicalAddress sourceText,
   ByteCount bufferSize,
   OptionBits convertFlags,
   UInt32 structureVersion,
   ByteCount *acceptedLength,
   LAMorphemesArrayPtr resultBuffer
);
```

Parameters

context

A reference to the language analysis context whose text you want to analyze. You can obtain a language analysis context by calling the function `LAOpenAnalysisContext`.

preferedEncoding

A value of type `TextEncoding` that specifies the encoding of text to use for both input and output. The text and length included in the `results` parameter are adjusted in accordance with the encoding specified here.

textLength

The length, in bytes, of the text you want to analyze.

sourceText

A pointer to the text you want to analyze.

bufferSize

The size of the buffer pointed to by the `resultBuffer` parameter.

convertFlags

An `OptionBits` value that specifies how to proceed with the analysis. Currently the only option you can set is `kLAEndOfSourceTextMask`. If this bit is set, the source text is analyzed to the end, and then results are generated. If this bit is not set, there is a possibility that the end portion of the source text is not yet analyzed when results are available. For example, when a large text file is analyzed it may be preferable to analyze it in chunks, returning results as each chunk is analyzed. You can specify this by passing 0 for the `convertFlags` parameter, advancing the analysis, and the setting `kLAEndOfSourceTextMask` when the whole file has been read.

structureVersion

The current version of `LAMorphemesArrayPtr`. You should pass `kLAMorphemesArrayVersion`.

acceptedLength

On output, the length of the source text that is accepted by the analysis engine.

resultBuffer

On output, a pointer to an array of `LAMorphmesArray` structures that contain the results of the morphological analysis.

Return Value

A result code. See "Result Codes" (page 1354).

Discussion

The function `LATextToMorphemes` analyzes the text specified in `textLength` and `sourceText`, and returns the results to `resultBuffer` in the form of `LAMorphemesArray`. While there are no restrictions on the length of text specified, the length in byte units of `sourceText` received in this call is set to `acceptedLength` at the point where the output buffer becomes full, or until all text provided has been analyzed. In practice, sections currently being analyzed exist within the analysis context, so be aware that the length returned may

not necessarily be the same as the section included in analysis results. This means that if the length of the returned text is shorter than the source text, analysis results are not complete. In this case, fetch the `results`, increment the `sourceText` by the `acceptedLength`, shorten `textLength` by the `acceptedLength`, and repeatedly call `LATextToMorphemes` until all the text is analyzed. The sample code below shows how to analyze text while loading it from a file.

```
while ( fileErr == noErr )
{
    fileErr = ReadFile (readBufferSize, &actualReadSize, readBuffer);
    if ( fileErr == eofErr )
        analyzeOption = kLAEndOfSourceTextMask;
    else
        analyzeOption = 0;
    analyzeLen = actualReadSize;
    analyzeText = readBuffer;
    result->morphemesCount = 0;
    while (analyzeLen || result->morphemesCount)
    {
        err = LATextToMorphemes (context, kTextEncodingMacJapanese,
                        analyzeLen, analyzeText, resultBufferSize,
                        analyzeOption, kLAMorphemesArrayVersion,
                        &acceptedLen, result);
        if (result->morphemesCount > 0 )
        {
                //
                // Retrieve result here...
                //
        }
        analyzeText += acceptedLen; // Increment source text ptr
        analyzeLen -= acceptedLen; // Decrement source text length
         }
}
```

If `kLAEndOfSourceTextMask` is specified and the analysis of all of the source text is done, the context becomes empty. If the analysis is suspended under this or other conditions (including errors), you must call the function `LAResetAnalysis` to clear the context.

Availability
Available in CarbonLib 1.0 and later when LanguageAnalysis 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

Data Types

HomographAccent

Defines a data type for a homographic accent.

```
typedef UInt8 HomographAccent;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

HomographDicInfoRec

Contains dictionary information for a homograph.

```
struct HomographDicInfoRec {
    DCMDictionaryID dictionaryID;
    DCMUniqueID uniqueID;
};
typedef struct HomographDicInfoRec HomographDicInfoRec;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

HomographWeight

Defines a data type for a homographic weighting value.

```
typedef UInt16 HomographWeight;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

JapanesePartOfSpeech

Defines a data type for a Japanese part of speech.

```
typedef MorphemePartOfSpeech JapanesePartOfSpeech;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAContextRef

A reference to an opaque language analysis context.

```
typedef struct OpaqueLAContextRef * LAContextRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAEnvironmentRef

A reference to an opaque language analysis environment structure.

```
typedef struct OpaqueLAEnvironmentRef * LAEnvironmentRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAHomograph

Defines a data types for a homograph node.

```
typedef AERecord LAHomograph;
```

Discussion
The Apple event record (`AERecord`) is the data type upon which many Language Analysis Manager data types are based. A homograph node is the minimum unit of analysis and is representative of an individual language. Typically a homograph node corresponds to one word obtained from the dictionary.

Homograph nodes include the character string which represents this language, but the content varies according to the type of analysis stipulated in the analysis environment. Depending on the type of environment, additional information may be included for a specific language.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAMorpheme

Defines a data type for a morpheme node.

```
typedef AERecord LAMorpheme;
```

Discussion
The Apple event record (`AERecord`) is the data type upon which many Language Analysis Manager data types are based. Morpheme nodes display the language of a specific part of speech for a particular text character strings, and have corresponding character string range, part of speech and homograph nodes within text character strings as attributes.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAMorphemeBundle

Defines a data type for a morpheme bundle.

```
typedef AERecord LAMorphemeBundle;
```

Discussion
The Apple event record (`AERecord`) is the data type upon which many Language Analysis Manager data types are based. Morpheme bundles are a collection of different solutions to morpheme analysis on one character string. The "different solutions" referred to here means that two solutions have different morpheme delimiters, or the same morpheme delimiters, but the parts of speech are not the same. Morpheme bundles have each of these different solutions in the from of a morpheme path. Morpheme bundles normally have multiple paths in the "most likely" order.

Within morpheme bundles, morpheme paths do not directly include morpheme nodes. Morpheme bundles have a list of morpheme nodes as one of their attributes distinct from the morpheme path, and morpheme paths have an index to that list. In this way, it is possible to share a morpheme node from one or more paths by indirectly indicating the morpheme node. In most cases, multiple paths within one bundle resemble one another to some extent, and multiple paths may be deemed to have the same morpheme node. One morpheme node may include many homograph nodes, making it bigger, so a mechanism such as this which allows sharing is important in maintaining a small data size.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAMorphemePath

Defines a data type for a morpheme path.

```
typedef AERecord LAMorphemePath;
```

Discussion
The Apple event record (`AERecord`) is the data type upon which many Language Analysis Manager data types are based. A morpheme path defines a single solution for the analysis of a morpheme. The path has an individual morphme delmiinter and part of speech.

There are two types of variation of morpheme paths which have a different way of holding the lower-place morpheme nodes, and in some cases they are used for different purposes. One is the morpheme path within the morpheme bundle mentioned earlier, where the path does not directly include morpheme nodes.

The other form is the morpheme path which can be used alone, and in this case, it is more convenient for it to be closed in that unit. If an application changes the operation of a morpheme node, the morpheme node must not be being shared. Therefore, for single morpheme paths, morpheme nodes are directly included in the morpheme path.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAMorphemeRec

Contains results of the analayis of one morpheme.

```
struct LAMorphemeRec {
    ByteCount sourceTextLength;
    LogicalAddress sourceTextPtr;
    ByteCount morphemeTextLength;
    LogicalAddress morphemeTextPtr;
    UInt32 partOfSpeech;
};
typedef struct LAMorphemeRec LAMorphemeRec;
```

Fields

`sourceTextLength`
The length of the source text for this morpheme.

`sourceTextPtr`
A pointer to the source text.

`morphemeTextLength`
The length of the result text for this morpheme.

`morphemeTextPtr`
A pointer to the result text.

`partOfSpeech`
The part of speech of this morpheme.

Discussion
This structure is an entry in the `LAMorphemesArray` data structure.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAMorphemesArray

Contains the results of high-level morphological analysis.

```
struct LAMorphemesArray {
    ItemCount morphemesCount;
    ByteCount processedTextLength;
    ByteCount morphemesTextLength;
    LAMorphemeRec morphemes[1];
};
typedef struct LAMorphemesArray LAMorphemesArray;
typedef LAMorphemesArray * LAMorphemesArrayPtr;
```

Fields

`morphemesCount`
The number of morphemes included.

`processedTextLength`
The processed source character length.

`morphemesTextLength`
The overall length of the result string.

`morphemes`
An array of morpheme records.

Discussion
When you perform high-level analysis, you can analyze stream-format text and obtain the results as an array of morpeme information.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAPropertyKey

Defines a data type for a language analysis property key.

```
typedef AEKeyword LAPropertyKey;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

LAPropertyType

Defines a data type for a language analysis property type.

```
typedef DescType LAPropertyType;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

MorphemePartOfSpeech

Defines a data type for a morpheme part of speech.

```
typedef UInt32 MorphemePartOfSpeech;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

MorphemeTextRange

Contains a range of text associated with a morpheme.

```
struct MorphemeTextRange {
    UInt32 sourceOffset;
    UInt32 length;
};
typedef struct MorphemeTextRange MorphemeTextRange;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`LanguageAnalysis.h`

Constants

File Creator Constants

Specify file creator for dictionary of Apple Japanese access methods.

```
enum {
    kAppleJapaneseDictionarySignature = 'jlan'
};
```

Analysis Engine Keywords

Specify analysis engine keywords for morpheme/homograph information.

```
enum {
    keyAEHomographDicInfo = 'lahd',
    keyAEHomographWeight = 'lahw',
    keyAEHomographAccent = 'laha'
};
```

Analysis Results Constants

Specify the nodes associated with analysis resutls.

```
enum {
    keyAELAMorphemeBundle = 'lmfb',
    keyAELAMorphemePath = 'lmfp',
    keyAELAMorpheme = 'lmfn',
    keyAELAHomograph = 'lmfh'
};
```

Morpheme Key Values

Specify key values used for morpheme/homgraph information.

```
enum {
    keyAEMorphemePartOfSpeechCode = 'lamc',
    keyAEMorphemeTextRange = 'lamt'
};
```

All Morphemes Constant

Specifies to use all morphemes.

```
enum {
    kLAAllMorphemes = 0
};
```

Leading and Trailing Constants

Specify constraints to apply to a string.

```
enum {
    kLADefaultEdge = 0,
    kLAFreeEdge = 1,
    kLAIncompleteEdge = 2
};
```

Converting Mask

Defines a mask for high-level API conversion flags.

```
enum {
    kLAEndOfSourceTextMask = 0x00000001
};
```

Morphemes Array Version

Specifies the version of the array used to hold morpheme analysis results.

```
enum {
    kLAMorphemesArrayVersion = 0
};
```

Conjugation Constants

Specify Japanese conjugations.

```
enum {
    kLASpeechKatsuyouGokan = 0x00000001,
    kLASpeechKatsuyouMizen = 0x00000002,
    kLASpeechKatsuyouRenyou = 0x00000003,
    kLASpeechKatsuyouSyuushi = 0x00000004,
    kLASpeechKatsuyouRentai = 0x00000005,
    kLASpeechKatsuyouKatei = 0x00000006,
    kLASpeechKatsuyouMeirei = 0x00000007
};
```

Parts of Speech Constants

Specify Japanese parts of speech.

```
enum {
    kLASpeechMeishi = 0x00000000,
    kLASpeechFutsuuMeishi = 0x00000000,
    kLASpeechJinmei = 0x00000100,
    kLASpeechJinmeiSei = 0x00000110,
    kLASpeechJinmeiMei = 0x00000120,
    kLASpeechChimei = 0x00000200,
    kLASpeechSetsubiChimei = 0x00000210,
    kLASpeechSoshikimei = 0x00000300,
    kLASpeechKoyuuMeishi = 0x00000400,
    kLASpeechSahenMeishi = 0x00000500,
    kLASpeechKeidouMeishi = 0x00000600,
    kLASpeechRentaishi = 0x00001000,
    kLASpeechFukushi = 0x00002000,
    kLASpeechSetsuzokushi = 0x00003000,
    kLASpeechKandoushi = 0x00004000,
    kLASpeechDoushi = 0x00005000,
    kLASpeechGodanDoushi = 0x00005000,
    kLASpeechKagyouGodan = 0x00005000,
    kLASpeechSagyouGodan = 0x00005010,
    kLASpeechTagyouGodan = 0x00005020,
    kLASpeechNagyouGodan = 0x00005030,
    kLASpeechMagyouGodan = 0x00005040,
    kLASpeechRagyouGodan = 0x00005050,
    kLASpeechWagyouGodan = 0x00005060,
    kLASpeechGagyouGodan = 0x00005070,
    kLASpeechBagyouGodan = 0x00005080,
    kLASpeechIchidanDoushi = 0x00005100,
    kLASpeechKahenDoushi = 0x00005200,
    kLASpeechSahenDoushi = 0x00005300,
    kLASpeechZahenDoushi = 0x00005400,
    kLASpeechKeiyoushi = 0x00006000,
    kLASpeechKeiyoudoushi = 0x00007000,
    kLASpeechSettougo = 0x00008000,
    kLASpeechSuujiSettougo = 0x00008100,
    kLASpeechSetsubigo = 0x00009000,
    kLASpeechJinmeiSetsubigo = 0x00009100,
    kLASpeechChimeiSetsubigo = 0x00009200,
    kLASpeechSoshikimeiSetsubigo = 0x00009300,
    kLASpeechSuujiSetsubigo = 0x00009400,
    kLASpeechMuhinshi = 0x0000A000,
    kLASpeechTankanji = 0x0000A000,
    kLASpeechKigou = 0x0000A100,
    kLASpeechKuten = 0x0000A110,
    kLASpeechTouten = 0x0000A120,
    kLASpeechSuushi = 0x0000A200,
    kLASpeechDokuritsugo = 0x0000A300,
    kLASpeechSeiku = 0x0000A400,
    kLASpeechJodoushi = 0x0000B000,
    kLASpeechJoshi = 0x0000C000
};
```

Parts of Speech Masks

Specify masks for parts of speech.

```
enum {
    kLASpeechRoughClassMask = 0x0000F000,
    kLASpeechMediumClassMask = 0x0000FF00,
    kLASpeechStrictClassMask = 0x0000FFF0,
    kLASpeechKatsuyouMask = 0x0000000F
};
```

Engine Limitations

Specify language analysis engine limitations.

```
enum {
    kMaxInputLengthOfAppleJapaneseEngine = 200
};
```

Analysis Engine Type Definitions

Specify language analysis engine type definitions for morpheme/homograph information.

```
enum {
    typeAEHomographDicInfo = 'lahd',
    typeAEHomographWeight = typeShortInteger,
    typeAEHomographAccent = 'laha'
};
```

Morpheme Types

Specify data types for morphemes.

```
enum {
    typeAEMorphemePartOfSpeechCode = 'lamc',
    typeAEMorphemeTextRange = 'lamt'
};
```

Morpheme Type Analysis Constants

Specify types used in morphological analysis.

```
enum { typeLAMorphemeBundle = typeAERecord, typeLAMorphemePath = typeAERecord,
typeLAMorpheme = typeAEList, typeLAHomograph = typeAEList };
```

Default Environment Names

Specify names for default environments for Japanese analysis.

```
#define kLAJapaneseKanaKanjiEnvironment "\pKanaKanjiConversion"
#define kLAJapaneseMorphemeAnalysisEnvironment
                                "\pJapaneseMorphemeAnalysis"
#define kLAJapaneseTTSEnvironment         "\pJapaneseTextToSpeech"
```

Result Codes

The most common result codes retuned by the Language Analysis Manager are listed in the table below.

Result Code	Value	Description
laTooSmallBufferErr	-6984	The output buffer is too small to store any result. Available in Mac OS X v10.0 and later.
laEnvironmentBusyErr	-6985	The specified environment is used. Available in Mac OS X v10.0 and later.
laEnvironmentNotFoundErr	-6986	The specified environment can't be found. Available in Mac OS X v10.0 and later.
laEnvironmentExistErr	-6987	An environment by the same name already exists. Available in Mac OS X v10.0 and later.
laInvalidPathErr	-6988	The path is not correct. Available in Mac OS X v10.0 and later.
laNoMoreMorphemeErr	-6989	There is nothing to read. Available in Mac OS X v10.0 and later.
laFailAnalysisErr	-6990	The analysis failed. Available in Mac OS X v10.0 and later.
laTextOverFlowErr	-6991	The text is too long. Available in Mac OS X v10.0 and later.
laDictionaryNotOpenedErr	-6992	The dictionary is not opened. Available in Mac OS X v10.0 and later.
laDictionaryUnknownErr	-6993	This dictionary can't be used with this environment. Available in Mac OS X v10.0 and later.
laDictionaryTooManyErr	-6994	There are too many dictionaries. Available in Mac OS X v10.0 and later.
laPropertyValueErr	-6995	Invalid property value. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
laPropertyUnknownErr	-6996	The property is unknown to this environment. Available in Mac OS X v10.0 and later.
laPropertyIsReadOnlyErr	-6997	The property is read only. Available in Mac OS X v10.0 and later.
laPropertyNotFoundErr	-6998	The property can't be found. Available in Mac OS X v10.0 and later.
laPropertyErr	-6999	There is an error in the property. Available in Mac OS X v10.0 and later.
laEngineNotFoundErr	-7000	The engine can't be found. Available in Mac OS X v10.0 and later.

Palette Manager Reference (Not Recommended)

Framework: ApplicationServices/ApplicationServices.h
Declared in Palettes.h

Overview

Important: The Palette Manager is deprecated as of Mac OS X v10.4. There is no replacement.

Prior to Mac OS X, applications could use the Palette Manager to ensure that the best set of colors is available when drawing to displays with limited color capabilities (pixel depth of 8 bits or less). For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Functions by Task

Function descriptions are grouped by programming task. For an alphabetical list of functions, see the API index.

Animating Palettes

`AnimateEntry` (page 1361) Deprecated in Mac OS X v10.4
Changes the color of a window's palette entry. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`AnimatePalette` (page 1362) Deprecated in Mac OS X v10.4
Changes the colors of a series of palette entries; it is similar to the `AnimateEntry` function, but it acts upon a range of entries. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Changing the Pixel Depth for a Video Device

`SetDepth` (page 1382)
Changes the pixel depth of a video device. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`HasDepth` (page 1371) Deprecated in Mac OS X v10.4
Determines whether a video device supports a specific pixel depth. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Drawing With Color Palettes

`PmBackColor` (page 1376) Deprecated in Mac OS X v10.4
Sets the background color field of the current graphics port to a palette color. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`PmForeColor` (page 1376) Deprecated in Mac OS X v10.4
Sets the foreground color field of the current graphics port to a palette color. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`RestoreBack` (page 1379) Deprecated in Mac OS X v10.4
Sets the current background color to the color you specify. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`RestoreFore` (page 1380) Deprecated in Mac OS X v10.4
Sets the current foreground color to the color you supply. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`SaveBack` (page 1381) Deprecated in Mac OS X v10.4
Saves the current background color. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`SaveFore` (page 1381) Deprecated in Mac OS X v10.4
Saves the current foreground color. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Initializing and Allocating Palettes

`DisposePalette` (page 1365) Deprecated in Mac OS X v10.4
Disposes of a palette. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`GetNewPalette` (page 1369) Deprecated in Mac OS X v10.4
Creates and initializes a palette from a `'pltt'` resource. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`NewPalette` (page 1373) Deprecated in Mac OS X v10.4
Allocates a new palette from colors in the color table. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Initializing the Palette Manager

`InitPalettes` (page 1372) Deprecated in Mac OS X v10.4
Initializes the Palette Manager. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`PMgrVersion` (page 1377) Deprecated in Mac OS X v10.4
Determines which version of the Palette Manager is executing; it returns an integer specifying the version number. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Interacting With the Window Manager

`ActivatePalette` (page 1360) Deprecated in Mac OS X v10.4
Changes the device color tables and generates window updates as needed to meet the color requirements of your window. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`GetPalette` (page 1370) Deprecated in Mac OS X v10.4
Obtains a window's palette. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`GetPaletteUpdates` (page 1371) Deprecated in Mac OS X v10.4
Obtains the update attribute of a palette. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`NSetPalette` (page 1374) Deprecated in Mac OS X v10.4
Associates a new palette with a window. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`SetPalette` (page 1385) Deprecated in Mac OS X v10.4
Associates a palette with a window. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`SetPaletteUpdates` (page 1386) Deprecated in Mac OS X v10.4
Sets the update attribute of a palette. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Manipulating Palette Entries

`Entry2Index` (page 1366) Deprecated in Mac OS X v10.4
Obtains the index for a specified entry in the current graphics port's palette on the current device. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`GetEntryColor` (page 1367) Deprecated in Mac OS X v10.4
Obtains the color of a palette entry. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`GetEntryUsage` (page 1367) Deprecated in Mac OS X v10.4
Obtains the usage and tolerance fields of a palette entry. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`SetEntryColor` (page 1383) Deprecated in Mac OS X v10.4
Changes the color of a palette entry. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`SetEntryUsage` (page 1384) Deprecated in Mac OS X v10.4
Modifies the usage category and tolerance values of a palette entry. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Manipulating Palettes and Color Tables

`CopyPalette` (page 1363) Deprecated in Mac OS X v10.4
: Copies entries from one palette to another. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`CTab2Palette` (page 1364) Deprecated in Mac OS X v10.4
: Copies the colors of a color table into a palette. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`Palette2CTab` (page 1375) Deprecated in Mac OS X v10.4
: Copies the colors of a palette into a color table. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`ResizePalette` (page 1378) Deprecated in Mac OS X v10.4
: Changes the size of a palette. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

`RestoreDeviceClut` (page 1379) Deprecated in Mac OS X v10.4
: Sets the color table of a graphics device to its default state. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Miscellaneous

`GetGray` (page 1368) Deprecated in Mac OS X v10.4
: Determines the best intermediate color between two colors on a given graphics device. (Deprecated. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Functions

ActivatePalette

Changes the device color tables and generates window updates as needed to meet the color requirements of your window. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void ActivatePalette (
   WindowRef srcWindow
);
```

Parameters

srcWindow
: A pointer to the window for which you want status changes reported.

Discussion

The Window Manager calls `ActivatePalette` when your window's status changes—for example, when your window opens, closes, moves, or becomes frontmost. You need to call the `ActivatePalette` function yourself if you change a palette—for example, by changing a color with the `SetEntryColor` function—and you want the changes to take place immediately, before the Window Manager would do it.

If the window specified in the `srcWindow` parameter is frontmost, `ActivatePalette` examines the information stored in the window's palette and attempts to provide the color environment described therein. It determines a list of devices on which to render the palette by intersecting the port rectangle of the window with each device. If the intersection is not empty and if the device has a color table, then `ActivatePalette` checks to see if the color environment is sufficient. If a change is required, `ActivatePalette` calls QuickDraw to reserve or modify the device's color entries as needed. The `ActivatePalette` function then generates update events for all windows that need color updates.

Calling `ActivatePalette` with an offscreen graphics world has no effect.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

AnimateEntry

Changes the color of a window's palette entry. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void AnimateEntry (
   WindowRef dstWindow,
   short dstEntry,
   const RGBColor *srcRGB
);
```

Parameters

dstWindow
A pointer to the window whose palette color is to be changed.

dstEntry
The palette entry to be changed.

srcRGB
A pointer to the new RGB value.

Discussion
The `AnimateEntry` function changes the RGB value of an animated entry for a window's palette. Each device for which that index has been reserved is immediately modified to contain the new value. This is not considered to be a change to the device's color environment because no other windows should be using the animated entry.

If the palette entry is not an animated color or if the associated indexes are no longer reserved, no animation occurs.

If you have blocked color updates in a window by using `SetPalette` with `cUpdates` set to `FALSE`, you may observe unintentional animation. This occurs when `ActivatePalette` reserves for animation device indexes that are already used in the window. Redrawing the window, which normally is the result of a color update event, removes any animated colors that do not belong to the window.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

AnimatePalette

Changes the colors of a series of palette entries; it is similar to the `AnimateEntry` function, but it acts upon a range of entries. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void AnimatePalette (
    WindowRef dstWindow,
    CTabHandle srcCTab,
    short srcIndex,
    short dstEntry,
    short dstLength
);
```

Parameters

dstWindow
A pointer to the window whose palette colors are to be changed.

srcCTab
A handle to the color table containing the new colors.

srcIndex
The source color table entry at which copying starts.

dstEntry
The palette entry at which copying starts.

dstLength
The number of palette entries to be changed.

Discussion
The `AnimatePalette` function changes the colors of a series of palette entries. Beginning at the index specified by the `srcIndex` parameter (which has a minimum value of 0), the number of entries specified in `dstLength` are copied from the source color table to the destination window's palette, beginning at the entry specified in the `dstEntry` parameter. If the source color table specified in `srcCTab` is not sufficiently large to accommodate the request, `AnimatePalette` modifies as many entries as possible and leaves the remaining entries unchanged.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

CopyPalette

Copies entries from one palette to another. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void CopyPalette (
   PaletteHandle srcPalette,
   PaletteHandle dstPalette,
   short srcEntry,
   short dstEntry,
   short dstLength
);
```

Parameters

srcPalette
A handle to the palette from which colors are copied.

dstPalette
A handle to the palette to which colors are copied.

srcEntry
The source palette entry at which copying starts.

dstEntry
The destination palette entry at which copying starts.

dstLength
The number of destination palette entries to change.

Discussion
The `CopyPalette` function copies entries from the source palette into the destination palette. The copy operation begins at the values specified by the `srcEntry` and `dstEntry` parameters, copying into as many entries as are specified by the `dstLength` parameter. `CopyPalette` resizes the destination palette when the number of entries after the copy operation is greater than it was before the copy operation.

`CopyPalette` does not call `ActivatePalette`, so your application is free to change the palette a number of times without causing a series of intermediate changes to the color environment. Your application should call `ActivatePalette` after completing all palette changes.

If either of the palette handles is `NULL`, `CopyPalette` does nothing.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

CTab2Palette

Copies the colors of a color table into a palette. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void CTab2Palette (
   CTabHandle srcCTab,
   PaletteHandle dstPalette,
   short srcUsage,
   short srcTolerance
);
```

Parameters

srcCTab
A handle to the color table whose colors are to be copied.

dstPalette
The palette to receive the colors.

srcUsage
A usage constant to be assigned the palette entries. Usage constants are described in "Update Constants" (page 1390).

srcTolerance
A tolerance value to be assigned the palette entries.

Discussion
The `CTab2Palette` function copies the fields from an existing color-table structure into an existing palette structure. If the structures are not the same size, then `CTab2Palette` resizes the palette structure to match the number of entries in the color-table structure. If the palette in `dstPalette` has any entries allocated for animation on any screen device, they are relinquished before the new colors are copied. The `srcUsage` and `srcTolerance` parameters are the value that you assign to the new colors.

If you want to use color-table animation, you can use `AnimateEntry` (page 1361) and `AnimatePalette` (page 1362) to change the colors in a palette and on corresponding devices. Changes made to a palette by `CTab2Palette` do not take effect until the next `ActivatePalette` function is performed. If either the color-table handle or the palette handle is `NULL`, `CTab2Palette` does nothing.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

DisposePalette

Disposes of a palette. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void DisposePalette (
   PaletteHandle srcPalette
);
```

Parameters
`srcPalette`
A handle to the palette to be disposed of.

Discussion
If the palette has any entries allocated for animation on any screen device, then `DisposePalette` relinquishes these entries before the palette's memory is released.

If a palette is attached to a window automatically—because the palette resource and the window have the same ID—you do not have to call the `DisposePalette` function to dispose of the function. The Palette Manager and Window Manager dispose of the palette automatically if the palette is replaced or if the window goes away.

However, if you explicitly attach a palette to a window with the `SetPalette` or `NSetPalette` function, your application owns the palette and is responsible for disposing of it. It is possible to attach a single palette to multiple windows; therefore, even when a window goes away and no longer needs a palette, other windows may still need it.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

Entry2Index

Obtains the index for a specified entry in the current graphics port's palette on the current device. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
SInt32 Entry2Index (
   short entry
);
```

Parameters

entry
The palette entry whose equivalent device index is to be returned.

Return Value
The index of the given `entry`.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

GetEntryColor

Obtains the color of a palette entry. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void GetEntryColor (
   PaletteHandle srcPalette,
   short srcEntry,
   RGBColor *dstRGB
);
```

Parameters

srcPalette

A handle to the palette to be accessed.

srcEntry

The palette entry whose color is desired.

dstRGB

A pointer to an RGB color structure to receive the palette color.

Discussion

You can modify the entry's color using the `SetEntryColor` (page 1383) function.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

GetEntryUsage

Obtains the usage and tolerance fields of a palette entry. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void GetEntryUsage (
   PaletteHandle srcPalette,
   short srcEntry,
   short *dstUsage,
   short *dstTolerance
);
```

Parameters

srcPalette

A handle to the palette to be accessed.

srcEntry

The palette entry whose usage and tolerance are desired.

dstUsage

A pointer to the usage value of the palette entry.

dstTolerance

A pointer to the tolerance value of the palette entry.

Discussion

You can modify the entry's usage and tolerance values by using the `SetEntryUsage` (page 1384) function.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

GetGray

Determines the best intermediate color between two colors on a given graphics device. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

```
Boolean GetGray (
   GDHandle device,
   const RGBColor *backGround,
   RGBColor *foreGround
);
```

Parameters

device

A handle to the graphics device for which an intermediate color or gray is needed.

backGround

The `RGBColor` structure for one of the two colors for which you want an intermediate color.

foreGround
On input, the `RGBColor` structure for the other of the two colors; upon completion, the best intermediate color between these two.

Return Value
If no gray is available (or if no distinguishable third color is available), the `foreGround` parameter is unchanged, and the function returns `FALSE`. If at least one gray or intermediate color is available, it is returned in the `foreGround` parameter, and the function returns `TRUE`.

Discussion
The `GetGray` function determines the midpoint values for the red, green, and blue values of the two colors you specify in the `backGround` and `foreGround` parameters.

You can also use `GetGray` to return the best gray. For example, when dimming an object, supply black and white as the two colors, and `GetGray` returns the best available gray that lies between them.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

GetNewPalette

Creates and initializes a palette from a `'pltt'` resource. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
PaletteHandle GetNewPalette (
   short PaletteID
);
```

Parameters
PaletteID
The resource ID of the source palette.

Return Value
A handle to the new palette.

Discussion
The `GetNewPalette` function detaches the resource when it creates the new palette, so you do not need to call the `ReleaseResource` function.

If you open a new color window with `GetNewCWindow`, the Window Manager calls `GetNewPalette` automatically, with `paletteID` equal to the window's resource ID. Therefore, if you have created a palette resource with the same ID as a window, the Window Manager and Palette Manager automatically create the palette for you and your application need not call `GetNewPalette` to create the palette.

To attach a palette to a window after creating it, use the `SetPalette` (page 1385) function. To change the entries in a palette after creating it, use the `SetEntryColor` (page 1383) and `SetEntryUsage` (page 1384) functions.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

GetPalette

Obtains a window's palette. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
PaletteHandle GetPalette (
   WindowRef srcWindow
);
```

Parameters

`srcWindow`
A pointer to the window for which you want the associated palette.

Return Value
A handle to the palette associated with the window specified in the `srcWindow` parameter or `NULL` if the window has no associated palette or is not a color window.

Discussion
Normally, the `GetPalette` function does not allocate memory, with one exception. When your application calls `GetPalette` to get a copy of the default application palette, the Palette Manager looks at the `AppPalette` global variable. If `AppPalette` is `NULL`, `GetPalette` makes a copy of the default system palette and returns a handle to this copy.

You request the default palette as follows:

myPaletteHndl = GetPalette ((WindowPtr) -1);

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

GetPaletteUpdates

Obtains the update attribute of a palette. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
short GetPaletteUpdates (
   PaletteHandle p
);
```

Parameters

p

A handle to the palette.

Return Value

One of the update attributes described in "Update Constants" (page 1390).

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

HasDepth

Determines whether a video device supports a specific pixel depth. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

```
short HasDepth (
   GDHandle gd,
   short depth,
   short whichFlags,
   short flags
);
```

Parameters

gd

A handle to the `GDevice` structure of the video device.

depth

The pixel depth for which you're testing.

whichFlags

The `gdDevType` constant, which represents a bit in the `gdFlags` field of the `GDevice` structure. If this bit is set to 0 in the `GDevice` structure, the video device is black and white; if the bit is set to 1, the device supports color.

flags

The value 0 or 1. If you pass 0 in this parameter, the `HasDepth` function tests whether the video device is black and white. If you pass 1 in this parameter, `HasDepth` tests whether the video device supports color.

Return Value

Returns 0 if the device does not support the depth you specify in the `depth` parameter or the display mode you specify in the `flags` parameter. Any other value indicates that the device supports the specified depth and display mode. The function result contains the mode ID that QuickDraw passes to the video driver to set its pixel depth and to specify color or black and white. You can pass this mode ID in the `depth` parameter for the `SetDepth` function to set the graphics device to the pixel depth and display mode for which you tested.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

InitPalettes

Initializes the Palette Manager. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void InitPalettes (
   void
);
```

Discussion
The `InitPalettes` function searches for devices that support a device color table and initializes an internal data structure for each one. Your application does not have to call `InitPalettes` because the Window Manager's `InitWindows` function calls it automatically.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

NewPalette

Allocates a new palette from colors in the color table. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
PaletteHandle NewPalette (
   short entries,
   CTabHandle srcColors,
   short srcUsage,
   short srcTolerance
);
```

Parameters

entries
The number of `ColorInfo` structures to be created in the new palette.

srcColors
A handle to the color table from which the colors are to be obtained. If no color table is provided (`srcColors` = `NULL`), then all colors in the palette are set to black (red, green, and blue equal to $0000).

srcUsage
The usage value to be assigned each `ColorInfo` structure in the palette.

srcTolerance
The tolerance value to be assigned each `ColorInfo` structure in the palette.

Return Value
A handle to the new palette.

Discussion
The `NewPalette` function fills the palette with as many RGB values from the color table as it has or can fit.

To attach a palette to a window after creating it, use the `SetPalette` (page 1385) function. To change the entries in a palette after creating it, use the `SetEntryColor` (page 1383) and `SetEntryUsage` (page 1384) functions.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

NSetPalette

Associates a new palette with a window. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void NSetPalette (
   WindowRef dstWindow,
   PaletteHandle srcPalette,
   short nCUpdates
);
```

Parameters

dstWindow
A pointer to the window to which you want to assign a new palette.

srcPalette
A handle to the palette you want to assign.

nCUpdates
An integer value in which you specify whether the window is to receive updates as a result of various changes to the color environment. See "Update Constants" (page 1390) for a description of the update options.

Discussion
`NSetPalette` changes the palette associated with the window specified in the `dstWindow` parameter to the palette specified by `srcPalette`. `NSetPalette` also records whether the window is to receive updates as a result of changes to its color environment. The update constants, which you pass to the `nCUpdates` parameter, determine when the window is updated.

This function is identical to the `SetPalette` (page 1385) function except that the `nCUpdates` parameter is an integer rather than a Boolean value, so that a variety of conditions can trigger an update event.

Use the SetPalette (page 1385) function if you do not need the flexibility that NSetPalette provides for update events.

Use the GetNewPalette (page 1369) function or the NewPalette (page 1373) function to create a new palette. To dispose of a palette, use the DisposePalette (page 1365) function.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Palettes.h

Palette2CTab

Copies the colors of a palette into a color table. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void Palette2CTab (
   PaletteHandle srcPalette,
   CTabHandle dstCTab
);
```

Parameters

srcPalette

A handle to the palette whose colors are to be used.

dstCTab

A handle to the color table to receive the colors.

Discussion

The Palette2CTab function copies all of the colors from an existing palette structure into an existing color-table structure. If the structures are not the same size, then Palette2CTab resizes the color-table structure to match the number of entries in the palette structure. If either the palette handle or the color-table handle is NULL, Palette2CTab does nothing.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

PmBackColor

Sets the background color field of the current graphics port to a palette color. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void PmBackColor (
   short dstEntry
);
```

Parameters

dstEntry

The palette entry whose color is to be used as the background color.

Discussion
The `PmBackColor` function sets the current color graphics port's `rgbBkColor` field to match the color in the entry specified by the `dstEntry` parameter of the palette associated with the current window structure. For courteous and tolerant entries, `PmBackColor` calls the `RGBBackColor` function using the RGB color of the palette entry. For animated colors, `PmBackColor` selects the recorded device index previously reserved for animation (if still present) and installs it in the color graphics port. The `rgbBgColor` field is set to the value from the palette entry. For explicit colors, `PmBackColor` places the value

```
dstEntry modulo (maxIndex +1)
```

into the color graphics port, where `maxIndex` is the largest index available in a device's color table. When multiple devices with different depths are present, `maxIndex` varies appropriately for each device.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

PmForeColor

Sets the foreground color field of the current graphics port to a palette color. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void PmForeColor (
   short dstEntry
);
```

Parameters

dstEntry

The palette entry whose color is to be used as the foreground color.

Discussion

The `PmForeColor` function sets the current color graphics port's `rgbFgColor` field to match the color in the entry specified by the `dstEntry` parameter of the palette associated with the current window structure. For courteous and tolerant entries, `PmForeColor` calls the `RGBForeColor` function using the RGB color of the palette entry. For animated colors, `PmForeColor` selects the recorded device index previously reserved for animation (if still present) and installs it in the color graphics port. The RGB foreground color field is set to the value from the palette entry. For explicit colors, `PmForeColor` places the value

```
dstEntry modulo (maxIndex +1)
```

into the color graphics port, where `maxIndex` is the largest index available in a device's color table. When multiple devices with different depths are present, the value of `maxIndex` varies appropriately for each device.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

PMgrVersion

Determines which version of the Palette Manager is executing; it returns an integer specifying the version number. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

```
short PMgrVersion (
   void
);
```

Return Value

`PMgrVersion` returns $0202 if system software version 7.0 is executing; $0201 if system software version 6.0.5 is executing; and $0200 if the original 32-Bit QuickDraw system extension is executing.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

ResizePalette

Changes the size of a palette. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void ResizePalette (
    PaletteHandle p,
    short size
);
```

Parameters

p

A handle to the palette to be resized.

size

The number of resulting entries in the palette.

Discussion

The `ResizePalette` function sets the palette specified in `srcPalette` to the number of entries indicated in the `size` parameter. If `ResizePalette` adds entries at the end of the palette, it sets them to `pmCourteous`, with the RGB values set to (0,0,0)—that is, black. If `ResizePalette` deletes entries from the end of the palette, it safely disposes of them.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

RestoreBack

Sets the current background color to the color you specify. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void RestoreBack (
   const ColorSpec *c
);
```

Parameters

c

A pointer to the `ColorSpec` structure containing the RGB color to be set as the background color. If you specify 0 in the `value` field of the `ColorSpec` structure, the `RestoreBack` function stores the RGB value in the `rgbFgColor` field of the current `CGrafPort` structure. If you specify 1 in the `value` field of the `ColorSpec` structure, the `RestoreBack` function stores the RGB value in the `pmBkColor` field of the `GrafVars` structure.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

RestoreDeviceClut

Sets the color table of a graphics device to its default state. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void RestoreDeviceClut (
   GDHandle gd
);
```

Parameters

gd

A handle to the `GDevice` structure. Pass `NULL` in the `gdh` parameter to restore all screens.

Discussion

The `RestoreDeviceClut` function changes the color table of the device specified by the `gdh` parameter to its default state. If this process changes any entries, the Palette Manager posts color updates to windows intersecting the device.

You do not need to use this function to restore the Finder's desktop colors, because its colors are automatically restored upon switching from applications that use the Palette Manager. Likewise, you need not worry when switching to another application that uses the Palette Manager. Although colors are not automatically restored in this case, if that application needs a certain set of colors, the Palette Manager provides them the moment that application comes to the front.

The reason to use `RestoreDeviceClut` is that you may be switching to an application that does not use the Palette Manager, in which case that application inherits your palette unless you restore the default color lookup tables for all the available display devices.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

RestoreFore

Sets the current foreground color to the color you supply. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void RestoreFore (
   const ColorSpec *c
);
```

Parameters

c

A pointer to the `ColorSpec` structure containing the RGB color to be set as the foreground color. If you specify 0 in the `value` field of the `ColorSpec` structure, the `RestoreFore` function stores the RGB value in the `rgbFgColor` field of the current `CGrafPort` structure. If you specify 1 in the `value` field of the `ColorSpec` structure, the `RestoreFore` function stores the RGB value in the `pmFgColor` field of the `GrafVars` structure.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Palettes.h`

SaveBack

Saves the current background color. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void SaveBack (
   ColorSpec *c
);
```

Parameters

c

A pointer to the `ColorSpec` structure to receive the current background color. A value of 0 in the `value` field of the `ColorSpec` structure specifies retrieving the RGB color from the `rgbBkColor` field of the `CGrafPort` structure; a value of 1 in the `value` field specifies retrieving the palette entry from the `pmBkColor` field of the `GrafVars` structure.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

SaveFore

Saves the current foreground color. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void SaveFore (
   ColorSpec *c
);
```

Parameters

c

A pointer to the `ColorSpec` structure to hold the current foreground color. A value of 0 in the `value` field of the `ColorSpec` structure specifies retrieving the RGB color from the `rgbFgColor` field of the `CGrafPort` structure; a value of 1 in the `value` field specifies retrieving the palette entry from the `pmFgColor` field of the `GrafVars` structure.On return, `ColorSpec` structure holds the current foreground color. You can save either QuickDraw's foreground color from the `CGrafPort` structure or the Palette Manager's foreground color from the `GrafVars` structure.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Palettes.h`

SetDepth

Changes the pixel depth of a video device. (**Deprecated.** There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

```
OSErr SetDepth (
   GDHandle gd,
   short depth,
   short whichFlags,
   short flags
);
```

Parameters

gd

A handle to the `GDevice` structure of the video device whose pixel depth you wish to change.

depth

The mode ID returned by the `HasDepth` (page 1371) function indicating that the video device supports the desired pixel depth. Alternatively, you can pass the desired pixel depth directly in this parameter, although you should use the `HasDepth` function to ensure that the device supports this depth.

whichFlags

The `gdDevType` constant, which represents a bit in the `gdFlags` field of the `GDevice` structure. (If this bit is set to 0 in the `GDevice` structure, the video device is black and white; if the bit is set to 1, the device supports color.

flags

The value 0 or 1. If you pass 0 in this parameter, the `SetDepth` function changes the video device to black and white; if you pass 1 in this parameter, `SetDepth` changes the video device to color.

Return Value
Returns zero if successful, or it returns a nonzero value if it cannot impose the desired depth and display mode on the requested device.

Discussion
The `SetDepth` function does not change the `'scrn'` resource; when the system is restarted, the original depth for this device is restored.

The Monitors control panel is the user interface for changing the pixel depth, color capabilities, and positions of video devices. Since the user can control the capabilities of the video device, your application should be flexible: although it may have a preferred pixel depth, your application should do its best to accommodate less than ideal conditions.

Use `SetDepth` only if your application can run on devices of a particular pixel depth and is unable to adapt to any other depth. Your application should display a dialog box that offers the user a choice between changing to that depth or canceling display of the image before your application uses `SetDepth`. Such a dialog box saves the user the trouble of going to the Monitors control panel before returning to your application.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.k.h`

SetEntryColor

Changes the color of a palette entry. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void SetEntryColor (
   PaletteHandle dstPalette,
   short dstEntry,
   const RGBColor *srcRGB
);
```

Parameters

dstPalette

The palette whose entry color is to be changed.

dstEntry

The palette entry to be changed.

srcRGB
A pointer to the new RGB color value.

Discussion
`SetEntryColor` marks the entry as having changed, but it does not change the color environment. That change occurs upon the next call to `ActivatePalette`. `SetEntryColor` marks modified entries such that the palette is updated, even though no update is required by a change in the color environment.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

SetEntryUsage

Modifies the usage category and tolerance values of a palette entry. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void SetEntryUsage (
   PaletteHandle dstPalette,
   short dstEntry,
   short srcUsage,
   short srcTolerance
);
```

Parameters
dstPalette
A handle to the palette to be modified.
dstEntry
The palette entry.
srcUsage
The new usage value; one or more usage constants.
srcTolerance
The new tolerance value.

Discussion
`SetEntryUsage` marks the entry as having changed, but it does not change the color environment. That change occurs upon the next call to `ActivatePalette`. Modified entries are marked such that the palette is updated even though no update is required by a change in the color environment. If either `srcUsage` or `srcTolerance` is set to $FFFF (–1), the entries are not changed.

This function allows you to easily modify a palette created with `NewPalette` or modified by `CTab2Palette`. For such palettes the `ciUsage` and `ciTolerance` fields of the `ColorInfo` structure are the same because you can designate only one value for each. You typically call `SetEntryUsage` after `NewPalette` or `CTab2Palette` to adjust and customize your palette.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

SetPalette

Associates a palette with a window. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void SetPalette (
   WindowRef dstWindow,
   PaletteHandle srcPalette,
   Boolean cUpdates
);
```

Parameters

dstWindow
A pointer to the window to which you want to assign a new palette.

srcPalette
A handle to the palette you want to assign.

cUpdates
A Boolean value in which your application specifies whether the window is to receive updates as a result of changes to the color environment. Specify `TRUE` if you want the window to be updated, even if the window is not the frontmost window. When a window is the frontmost window, changes to its palette cause it to get an update event regardless of how the `cUpdates` parameter is set.

Discussion
You can use the `NSetPalette` (page 1374) function, which does the same thing as `SetPalette`, when you need greater flexibility in setting criteria for updates. The `nCUpdates` parameter for the `NSetPalette` function includes the option of turning off updates when the window is the frontmost window.

Use the `NSetPalette` function to associate a palette with a window but with additional options as to when an update event is triggered by changes to the color environment.

Use the `GetNewPalette` (page 1369) function or the `NewPalette` (page 1373) function to create a new palette. To dispose of a palette, use the `DisposePalette` (page 1365) function.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

SetPaletteUpdates

Sets the update attribute of a palette. (Deprecated in Mac OS X v10.4. There is no replacement; 8-bit graphics mode is not supported by the Mac OS X GUI.)

Not recommended.

```
void SetPaletteUpdates (
   PaletteHandle p,
   short updates
);
```

Parameters

p
A handle to the palette.

updates
One of the update attributes for the `NSetPalette` function. See "Update Constants" (page 1390) for a description of the update attributes.

Special Considerations

For applications running in Mac OS X, the Palette Manager is no longer relevant because display devices always support direct color (pixel depth of 16 or 32 bits). The palette-based graphics model only works in 256-color (8-bit pseudocolor) modes of operation, which are not supported for the Mac OS X GUI.

There is some support for palettes in Quartz Services; see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Palettes.h`

Data Types

ColorInfo

Specifies color information for each color in a palette.

```
struct ColorInfo {
    RGBColor ciRGB;
    short ciUsage;
    short ciTolerance;
    short ciDataFields[3];
};
typedef struct ColorInfo ColorInfo;
typedef ColorInfo * ColorInfoPtr;
```

Fields

`ciRGB`
: An RGB color value, which is defined by the `RGBColor` structure. It contains three fields that contain integer values for defining, respectively, the red, green, and blue values of the color.

`ciUsage`
: One or more of the usage constants, specifying how this entry is to be used. The `ciUsage` field can contain any of the `Usage Constants` (page 1388).

`ciTolerance`
: An integer expressing the range in RGB space within which the red, green, and blue values must fall to satisfy this entry. A tolerance value of $0000 means that only an exact match is acceptable. Values of $0xxx other than $0000 are reserved and should not be used in applications.

`ciDataFields`
: Private fields.

Discussion

Each color information structure in a palette comprises an RGB color value, information describing how the color is to be used, a tolerance value for colors that need only be approximated, and private fields. You should not create and modify the public fields directly; instead, use Palette Manager functions such as `SetEntryColor` and `SetEntryUsage`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Palettes.h`

Palette

Represents a set of colors optimized for use on display devices with a limited number of colors.

```
struct Palette {
    short pmEntries;
    short pmDataFields[7];
    ColorInfo pmInfo[1];
};
typedef struct Palette Palette;
typedef Palette * PalettePtr;
```

Fields

`pmEntries`

The number of `ColorInfo` structures in the `pmInfo` array.

`pmDataFields`

Private fields used by the Palette Manager.

`pmInfo`

An array of `ColorInfo` (page 1387) structures.

Discussion

A palette structure contains a header and a collection of color information structures, one for each color in the palette.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Palettes.h`

Constants

Usage Constants

Constants used to determine how the Palette Manager uses a specific palette color.

```
enum {
    pmCourteous = 0,
    pmDithered = 0x0001,
    pmTolerant = 0x0002,
    pmAnimated = 0x0004,
    pmExplicit = 0x0008,
    pmWhite = 0x0010,
    pmBlack = 0x0020,
    pmInhibitG2 = 0x0100,
    pmInhibitC2 = 0x0200,
    pmInhibitG4 = 0x0400,
    pmInhibitC4 = 0x0800,
    pmInhibitG8 = 0x1000,
    pmInhibitC8 = 0x2000
};
```

Constants

`pmCourteous`
The color accepts whatever value the Color Manager determines to be the closest match currently available in the device color table.

Available in Mac OS X v10.0 and later.

Declared in `Palettes.h`.

`pmDithered`
[description forthcoming]

Available in Mac OS X v10.0 and later.

Declared in `Palettes.h`.

`pmTolerant`
The color accepts the Color Manager's choices on an indexed device.

Available in Mac OS X v10.0 and later.

Declared in `Palettes.h`.

`pmAnimated`
The color is used for special color animation effects.

Available in Mac OS X v10.0 and later.

Declared in `Palettes.h`.

`pmExplicit`
The color specifies an index value rather than an RGB color and always generates the corresponding entry from the device's color table.

Available in Mac OS X v10.0 and later.

Declared in `Palettes.h`.

`pmWhite`
Assign color to white on a 1-bit device.

Available in Mac OS X v10.0 and later.

Declared in `Palettes.h`.

`pmBlack`
Assign color to black on a 1-bit device.

Available in Mac OS X v10.0 and later.

Declared in `Palettes.h`.

`pmInhibitG2`
Inhibit on 2-bit grayscale device.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmInhibitC2`
Inhibit on 2-bit color device.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmInhibitG4`
Inhibit on 4-bit grayscale device.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmInhibitC4`
Inhibit on 4-bit color device.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmInhibitG8`
Inhibit on 8-bit grayscale device.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmInhibitC8`
Inhibit on 8-bit color device.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

Discussion
You can logically `AND` these constants in certain combinations to specify the value of `ciUsage` in a `ColorInfo` (page 1387) record.

The inhibit constants are used to indicate that a graphics device should be prevented from displaying the color at specified pixel depths. They are always used in combination with other usage constants.

Update Constants

Constants used to determine whether a window is updated based on various changes to the color environment.

```
enum {
    pmNoUpdates = 0x8000,
    pmBkUpdates = 0xA000,
    pmFgUpdates = 0xC000,
    pmAllUpdates = 0xE000
};
```

Constants

`pmNoUpdates`
Do not update the window when its color environment changes.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmBkUpdates`
Update the window only when it is not the active window.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmFgUpdates`
Update the window only when it is the active window.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

`pmAllUpdates`
Update the window whenever its color environment changes.
Available in Mac OS X v10.0 and later.
Declared in `Palettes.h`.

Discussion

You use these constants in the `nCUpdates` parameter of the `NSetPalette` (page 1374) function and the `updates` parameter of the `SetPaletteUpdates` (page 1386) function.

Pasteboard Manager Reference

Framework:	HIServices/HIServices.h
Declared in	Pasteboard.h
Companion guide	Pasteboard Manager Programming Guide

Overview

The Pasteboard Manager lets you create and manipulate pasteboards, which act as holding containers for data to be transferred from one place to another. Typically you use pasteboards to facilitate copy-and-paste or drag-and-drop operations, but you can use them whenever you need to temporarily store data that will be moved elsewhere.

Note: The Pasteboard Manager supersedes the Scrap Manager and the drag flavor APIs in the Drag Manager, adding greater flexibility in the type and quantity of data to be transferred. Pasteboard Manager pasteboards are also fully compatible with Cocoa pasteboards.

Functions by Task

Creating and Using Pasteboards

`PasteboardCreate` (page 1397)
Creates a reference to the specified global pasteboard.

`PasteboardSynchronize` (page 1402)
Synchronizes the local pasteboard reference to reflect the contents of the global pasteboard.

`PasteboardClear` (page 1394)
Clears the contents of the specified pasteboard.

`PasteboardCopyName` (page 1396)
Gets the name of a pasteboard.

`PasteboardGetItemCount` (page 1398)
Obtains the number of data items in the specified pasteboard.

`PasteboardGetItemIdentifier` (page 1399)
Obtains the item identifier for an item in a pasteboard.

Manipulating Pasteboard Flavor Data

`PasteboardCopyItemFlavors` (page 1395)
Obtains an array of flavors for a specified item in a pasteboard.

`PasteboardGetItemFlavorFlags` (page 1398)
Obtains the flags for a given flavor.

`PasteboardCopyItemFlavorData` (page 1395)
Obtains data from a pasteboard for the desired flavor.

`PasteboardPutItemFlavor` (page 1399)
Adds flavor data or a promise to the specified pasteboard.

`PasteboardSetPromiseKeeper` (page 1402)
Registers the promise keeper callback function for a pasteboard.

`PasteboardCopyPasteLocation` (page 1396)
Determines the location in which to paste promised data.

`PasteboardSetPasteLocation` (page 1401)
Sets the paste location before requesting flavor data.

`PasteboardResolvePromises` (page 1400)
Resolves all promises to a given pasteboard.

Functions

PasteboardClear

Clears the contents of the specified pasteboard.

```
OSStatus PasteboardClear (
   PasteboardRef inPasteboard
);
```

Parameters

inPasteboard
The pasteboard you want to clear.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
After calling this function, the application owns the pasteboard and can add data to it. You must call `PasteboardClear` before modifying a pasteboard.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CarbonSketch

Declared In
`Pasteboard.h`

PasteboardCopyItemFlavorData

Obtains data from a pasteboard for the desired flavor.

```
OSStatus PasteboardCopyItemFlavorData (
   PasteboardRef inPasteboard,
   PasteboardItemID inItem,
   CFStringRef inFlavorType,
   CFDataRef *outData
);
```

Parameters

inPasteboard
The pasteboard containing the data.

inItem
The identifier for the item whose flavor data you want to obtain.

inFlavorType
The flavor of the data you want to obtain, specified as a uniform type identifier.

outData
On return, `outData` points to the flavor data. You must release this data using `CFRelease` when you are done using it.

Return Value

A result code. See "Pasteboard Manager Result Codes" (page 1407).

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CarbonCocoa_PictureCursor
CarbonSketch

Declared In

`Pasteboard.h`

PasteboardCopyItemFlavors

Obtains an array of flavors for a specified item in a pasteboard.

```
OSStatus PasteboardCopyItemFlavors (
   PasteboardRef inPasteboard,
   PasteboardItemID inItem,
   CFArrayRef *outFlavorTypes
);
```

Parameters

inPasteboard
The pasteboard containing the data.

inItem
The identifier for the item whose flavors you want to obtain.

outFlavorTypes
On return, `outFlavorTypes` points to an array of flavors, specified as uniform type identifiers. You must release this array by calling `CFRelease` when you are done using it.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CarbonSketch

Declared In
`Pasteboard.h`

PasteboardCopyName

Gets the name of a pasteboard.

```
OSStatus PasteboardCopyName (
    PasteboardRef inPasteboard,
    CFStringRef *outName
);
```

Parameters

inPasteboard
The pasteboard whose name you want to retrieve.

outName
A pointer to a `CFStringRef` variable allocated by the caller. On return, the string contains the name of the specified pasteboard. The caller is responsible for releasing the string.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
You can use this function to discover the name of a uniquely-named pasteboard so that other processes may access it.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Pasteboard.h`

PasteboardCopyPasteLocation

Determines the location in which to paste promised data.

```
OSStatus PasteboardCopyPasteLocation (
   PasteboardRef inPasteboard,
   CFURLRef *outPasteLocation
);
```

Parameters

inPasteboard
The pasteboard whose paste location you want to determine.

outPasteLocation

On return, `outPasteLocation` points to a CFURL indicating the paste location. You must release this URL when you are done with it by calling `CFRelease`.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
You typically call this function from your promise keeper callback function to determine where to deliver a promised file. It is also used to tell a translation service where to place a translated file.

This function is analogous to the Drag Manager function `GetDropLocation`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Pasteboard.h`

PasteboardCreate

Creates a reference to the specified global pasteboard.

```
OSStatus PasteboardCreate (
   CFStringRef inName,
   PasteboardRef *outPasteboard
);
```

Parameters

inName

The name of the pasteboard to reference. To create a new pasteboard, specify a unique name. Pasteboard names should have a reverse DNS-style name to ensure uniqueness. You may also pass one of the following constants:

- `kPasteboardFind` (page 1405) to obtain a reference to the global Find pasteboard
- `kPasteboardClipboard` (page 1405) to obtain a reference to the global Clipboard
- `kPasteboardUniqueName` (page 1405) to ask the Pasteboard Manager to create a new pasteboard with a unique name

outPasteboard

A pointer to a `PasteboardRef` variable allocated by the caller. On return, the variable refers to the pasteboard specified in the *inName* parameter.

When you are finished using the pasteboard, you can call `CFRelease` to release the reference. If you do not release the reference, the pasteboard continues to exist even after your application terminates. A subsequently launched application can find a previously created pasteboard by name and examine the data within it.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
You can use this function to create a reference to a new or existing pasteboard.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CarbonSketch

Declared In
`Pasteboard.h`

PasteboardGetItemCount

Obtains the number of data items in the specified pasteboard.

```
OSStatus PasteboardGetItemCount (
   PasteboardRef inPasteboard,
   ItemCount *outItemCount
);
```

Parameters

`inPasteboard`
The pasteboard whose items you want to count.

`outItemCount`
On return, `outItemCount` points to the number of items in the pasteboard.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
You usually call this function before calling `PasteboardGetItemIdentifier` (page 1399).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CarbonSketch

Declared In
`Pasteboard.h`

PasteboardGetItemFlavorFlags

Obtains the flags for a given flavor.

```
OSStatus PasteboardGetItemFlavorFlags (
   PasteboardRef inPasteboard,
   PasteboardItemID inItem,
   CFStringRef inFlavorType,
   PasteboardFlavorFlags *outFlags
);
```

Parameters

`inPasteboard`
The pasteboard containing the data.

`inItem`
The identifier for the item whose flavor flags you want to obtain.

inFlavorType
The flavor whose flags you want to obtain.

outFlags
On return, `outFlags` points to a bit field containing the flavor flags. See "Pasteboard Flavor Flags" (page 1405) for a list of possible values.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Pasteboard.h`

PasteboardGetItemIdentifier

Obtains the item identifier for an item in a pasteboard.

```
OSStatus PasteboardGetItemIdentifier (
   PasteboardRef inPasteboard,
   CFIndex inIndex,
   PasteboardItemID *outItem
);
```

Parameters

inPasteboard
The pasteboard containing the data.

inIndex
The one-based index number of the data item whose identifier you want to obtain.

outItem
On return, `outItem` points to the item identifier for the data item at index `inIndex`.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
The item index is one-based to match the convention used by the Drag Manager.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CarbonCocoa_PictureCursor
CarbonSketch

Declared In
`Pasteboard.h`

PasteboardPutItemFlavor

Adds flavor data or a promise to the specified pasteboard.

```
OSStatus PasteboardPutItemFlavor (
   PasteboardRef inPasteboard,
   PasteboardItemID inItem,
   CFStringRef inFlavorType,
   CFDataRef inData,
   PasteboardFlavorFlags inFlags
);
```

Parameters

inPasteboard

The pasteboard to which to add flavor data or a promise.

inItem

The identifier for the item to which to add flavor data or a promise.

inFlavorType

The flavor type of the data or promise you are adding, specified as a uniform type identifier.

inData

The data to add, or a promise to supply the data later. If you pass a CFData object, you may safely release the object when this function returns. To indicate that the data is promised, pass `kPasteboardPromisedData` (page 1407). For more information about promised data, see the Discussion below.

inFlags

A bit field of flags for the specified flavor.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
If you promise data, you must implement a promise keeper callback to deliver the data when asked for it (see `PasteboardPromiseKeeperProcPtr` (page 1403) for more details). Typically, you store promises instead of the actual data when the data requires a large overhead to generate. You can call this function multiple times to add multiple flavors to a given item. You should add the flavors in your application's order of preference or richness.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CarbonSketch

Declared In
`Pasteboard.h`

PasteboardResolvePromises

Resolves all promises to a given pasteboard.

```
OSStatus PasteboardResolvePromises (
   PasteboardRef inPasteboard
);
```

Parameters

inPasteboard

The pasteboard whose promises your application wants to resolve. If you pass `kPasteboardResolveAllPromises` (page 1407), all the promises for all pasteboards handled by the application are resolved.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
You should call this function when your application quits or otherwise becomes unavailable.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Pasteboard.h`

PasteboardSetPasteLocation

Sets the paste location before requesting flavor data.

```
OSStatus PasteboardSetPasteLocation (
   PasteboardRef inPasteboard,
   CFURLRef inPasteLocation
);
```

Parameters

inPasteboard

The pasteboard you want to obtain flavor data from.

inPasteLocation

A Core Foundation URL indicating where to put the data.

Return Value
A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion
Applications that receive pasteboard data in the form of a file should call this function before calling `PasteboardCopyItemFlavorData` (page 1395), so the sender knows where to place the promised data. If the requested data was promised by the sender, the sending application calls `PasteboardCopyPasteLocation` (page 1396) to determine where to put it.

If a paste location is not applicable for your application, you don't need to call this function.

This function is analogous to the Drag Manager function `SetDropLocation`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Pasteboard.h`

PasteboardSetPromiseKeeper

Registers the promise keeper callback function for a pasteboard.

```
OSStatus PasteboardSetPromiseKeeper (
   PasteboardRef inPasteboard,
   PasteboardPromiseKeeperProcPtr inPromiseKeeper,
   void *inContext
);
```

Parameters

inPasteboard

The pasteboard to assign the promise keeper callback. If you have multiple pasteboards, you can assign multiple callbacks.

inPromiseKeeper

A pointer to your promise keeper callback function. See `PasteboardPromiseKeeperProcPtr` (page 1403) for more information about implementing the callback.

inContext

A pointer to application-defined data. The value you pass in this parameter is passed to your promise keeper callback function when it is called.

Return Value

A result code. See "Pasteboard Manager Result Codes" (page 1407).

Discussion

You must have registered a promise keeper callback function before promising any data on the specified pasteboard.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Pasteboard.h`

PasteboardSynchronize

Synchronizes the local pasteboard reference to reflect the contents of the global pasteboard.

```
PasteboardSyncFlags PasteboardSynchronize (
   PasteboardRef inPasteboard
);
```

Parameters

inPasteboard

The pasteboard you want to synchronize.

Return Value

A flag indicating what synchronization actions occurred.

Discussion

Calling this function compares the local pasteboard reference with its global pasteboard. If the global pasteboard was modified, it updates the local pasteboard reference to reflect this change. You typically call this function whenever your application becomes active, so that its pasteboard information reflects any changes that occurred while it was in the background. This function has low overhead, so you should call it whenever you suspect a global pasteboard may have been changed.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CarbonSketch

Declared In
`Pasteboard.h`

Callbacks

PasteboardPromiseKeeperProcPtr

Defines a pointer to a callback function that supplies the actual data promised on the pasteboard.

```
typedef OSStatus (*PasteboardPromiseKeeperProcPtr)
(
PasteboardRef pasteboard,
PasteboardItemID item,
CFStringRef flavorType,
void* context
);
```

You would declare your promise keeper callback function named `MyPromiseKeeper` like this:

```
OSStatus MyPromiseKeeper (
PasteboardRef pasteboard,
PasteboardItemID item,
CFStringRef flavorType,
void* context
);
```

Parameters

pasteboard
The pasteboard whose promise you need to fulfill.

item
The pasteboard item identifier containing the promised flavor.

flavorType
The flavor of the data being requested, specified as a uniform type identifier.

context
The pointer you passed to `PasteboardSetPromiseKeeper` (page 1402) in the `inContext` parameter.

Discussion
When promising any flavor data on a pasteboard using `PasteboardPutItemFlavor` (page 1399), you must implement a callback function to fulfill that promise.

If your application delivers the promised data as a file, your callback should call `PasteboardCopyPasteLocation` (page 1396) to determine where to deliver the requested data and then take the necessary steps to do so.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Pasteboard.h`

Data Types

PasteboardRef

Defines an opaque type that represents a pasteboard.

```
typedef struct OpaquePasteboardRef *PasteboardRef;
```

Discussion
This is a Core Foundation type. For more information, see *CFType Reference*.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Pasteboard.h`

PasteboardItemID

Defines a pasteboard item identifier.

```
typedef void* PasteboardItemID;
```

Discussion
You can use any arbitrary (but unique) ID to identify your pasteboard items when placing them on a pasteboard. For example, you may want to identify items by their internal memory address. Only the owning application should interpret its pasteboard item identifiers.

When your application's promise keeper callback function is invoked, it receives a pasteboard item ID, and your application must be able to map that ID to the corresponding promised data.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Pasteboard.h`

Constants

Pasteboard Name Constants

Define the pasteboard names used when calling `PasteboardCreate` (page 1397).

```
#define kPasteboardClipboard CFSTR("com.apple.pasteboard.clipboard")
#define kPasteboardFind CFSTR("com.apple.pasteboard.find")
#define kPasteboardUniqueName (CFStringRef)NULL
```

Constants

`kPasteboardClipboard`
The global Clipboard (used for copy and paste).

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

`kPasteboardFind`
The global Find pasteboard (used for search fields).

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

`kPasteboardUniqueName`
A system-declared pasteboard that is guaranteed to be unique. Each time you call `PasteboardCreate` (page 1397) with this constant, you create a different, unique pasteboard.

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

Pasteboard Flavor Flags

Indicate useful information associated with pasteboard item flavors.

```
enum {
   kPasteboardFlavorNoFlags = 0,
   kPasteboardFlavorSenderOnly = (1 << 0),
   kPasteboardFlavorSenderTranslated = (1 << 1),
   kPasteboardFlavorNotSaved = (1 << 2),
   kPasteboardFlavorRequestOnly = (1 << 3),
   kPasteboardFlavorSystemTranslated = (1 << 8),
   kPasteboardFlavorPromised = (1 << 9)
};
typedef UInt32 PasteboardFlavorFlags;
```

Constants

`kPasteboardFlavorNoFlags`
No flag information exists for this flavor.

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

`kPasteboardFlavorSenderOnly`
Only the process that added this flavor can see it. Typically used to tag proprietary data that can be cut, dragged, or pasted only within the owning application. This flag is identical to the Drag Manager `flavorSenderOnly` flag.

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

`kPasteboardFlavorSenderTranslated`
: The sender translated this data in some fashion before adding it to the pasteboard. The Finder cannot store flavor items marked with this flag in clipping files. This flag is identical to the Drag Manager `flavorSenderTranslated` flag.

  Available in Mac OS X v10.3 and later.

  Declared in `Pasteboard.h`.

`kPasteboardFlavorNotSaved`
: The receiver should not save the data provided for this flavor. The data may become stale after a drag, or the flavor may be used only to augment another flavor. For example, an application may add a flavor whose only purpose is to supply additional information (say text style) about another flavor. The Finder cannot store flavor items marked with this flag in clipping files. This flag is identical to the Drag Manager `flavorNotSaved` flag.

  Available in Mac OS X v10.3 and later.

  Declared in `Pasteboard.h`.

`kPasteboardFlavorRequestOnly`
: When the sender sets this flag, this flavor is hidden from calls to `PasteboardCopyItemFlavors` (page 1395). However, you can obtain the flavor flags and data by calling `PasteboardGetItemFlavorFlags` (page 1398) or `PasteboardCopyItemFlavorData` (page 1395). The net result is that applications cannot obtain this flavor unless they are already aware it exists and specifically request it. This functionality can be useful for copying and pasting proprietary data within a suite of applications.

  Available in Mac OS X v10.3 and later.

  Declared in `Pasteboard.h`.

`kPasteboardFlavorSystemTranslated`
: This data flavor is available through the Translation Manager. That is, the Translation Manager must translate the sender's data before the receiver can receive it. This flag is set automatically when appropriate and cannot be set programmatically. The Finder cannot store flavor items marked with this flag in clipping files. This flag is identical to the Drag Manager `flavorSystemTranslated` flag.

  Available in Mac OS X v10.3 and later.

  Declared in `Pasteboard.h`.

`kPasteboardFlavorPromised`
: The data associated with this flavor is not yet on the pasteboard. Typically data is promised to the pasteboard if it requires a lot of overhead to generate. If the receiver requests the data, the sender is notified so it can supply the promised data. This flag is set automatically when appropriate and cannot be set programmatically. This flag is identical to the Drag Manager `flavorDataPromised` flag.

  Available in Mac OS X v10.3 and later.

  Declared in `Pasteboard.h`.

Pasteboard Synchronization Flags

Indicate the pasteboard status after a call to `PasteboardSynchronize` (page 1402).

```
enum {
   kPasteboardModified = (1 << 0),
   kPasteboardClientIsOwner = (1 << 1)
};
typedef UInt32 PasteboardSyncFlags;
```

Constants

`kPasteboardModified`

The pasteboard was modified since the last time the application accessed it, and the local pasteboard has been synchronized to reflect any changes. Your application should check to see what flavors are now available on the pasteboard and update appropriately (for example, enabling the Paste menu item).

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

`kPasteboardClientIsOwner`

The application recently cleared the pasteboard and is its current owner. The application can add flavor data to the pasteboard as desired.

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

Pasteboard Promise Constants

Define constants related to the use of promised data.

```
#define kPasteboardPromisedData (CFDataRef)NULL
#define kPasteboardResolveAllPromises (PasteboardRef)NULL
```

Constants

`kPasteboardPromisedData`

Indicates a promise that pasteboard data will be supplied later. Used when calling `PasteboardPutItemFlavor` (page 1399).

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

`kPasteboardResolveAllPromises`

Indicates that all promises on all global pasteboard resources owned by this application should be resolved. Used when calling `PasteboardResolvePromises` (page 1400).

Available in Mac OS X v10.3 and later.

Declared in `Pasteboard.h`.

Result Codes

The table below lists the most common error codes returned by the Pasteboard Manager.

Result Code	Value	Description
badPasteboardSyncErr	-25130	The pasteboard has been modified and must be synchronized before use. Available in Mac OS X v10.3 and later.
badPasteboardIndexErr	-25131	The specified pasteboard item index does not exist. Available in Mac OS X v10.3 and later.
badPasteboardItemErr	-25132	The item reference does not exist. Available in Mac OS X v10.3 and later.
badPasteboardFlavorErr	-25133	The item flavor does not exist. Available in Mac OS X v10.3 and later.
duplicatePasteboardFlavorErr	-25134	The item flavor already exists. Available in Mac OS X v10.3 and later.
notPasteboardOwnerErr	-25135	The application did not clear the pasteboard before attempting to add flavor data. Available in Mac OS X v10.3 and later.
noPasteboardPromiseKeeperErr	-25136	The application attempted to add promised data without previously registering a promise keeper callback. Available in Mac OS X v10.3 and later.

Picture Utilities Reference (Not Recommended)

Framework: ApplicationServices/ApplicationServices.h

Declared in PictUtils.h

Overview

Important: The Picture Utilities are deprecated as of Mac OS X v10.4. The replacement API for all QuickDraw technologies is Quartz 2D (Core Graphics). See *Quartz Programming Guide for QuickDraw Developers* for strategies to replace QuickDraw code with Quartz 2D..

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

QuickDraw pictures are sequences of saved drawing commands. Pictures provide a common medium for the sharing of image data.

The Picture Utilities allow your application to gather information about a picture, such as color, fonts, picture comments, and resolution. You can also use the Picture Utilities to gather information about the colors in pixel maps.

Functions by Task

Collecting Picture Information

`DisposePictInfo` (page 1412) Deprecated in Mac OS X v10.4

Disposes of the private data structures allocated by the `NewPictInfo` function. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPictInfo` (page 1414) Deprecated in Mac OS X v10.4

Gathers information about a single picture. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPixMapInfo` (page 1416) Deprecated in Mac OS X v10.4

Gathers color information about a single pixel map or bitmap. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewPictInfo` (page 1422) Deprecated in Mac OS X v10.4
Begins collecting pictures, pixel maps, and bitmaps for a survey of pictures. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RecordPictInfo` (page 1424) Deprecated in Mac OS X v10.4
Adds a picture to an informational survey of multiple pictures. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RecordPixMapInfo` (page 1424) Deprecated in Mac OS X v10.4
Adds a pixel map or bitmap to an informational survey of multiple pixel maps and bitmaps. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RetrievePictInfo` (page 1425) Deprecated in Mac OS X v10.4
Returns information about all the pictures, pixel maps, and bitmaps included in a survey. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Using Universal Procedure Pointers

`DisposeCalcColorTableUPP` (page 1411) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a color table calculation callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeDisposeColorPickMethodUPP` (page 1411) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a method disposal callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeInitPickMethodUPP` (page 1412) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a method initialization callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeRecordColorsUPP` (page 1413) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a color recording callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvokeCalcColorTableUPP` (page 1418) Deprecated in Mac OS X v10.4
Invokes a color table calculation callback, using a universal procedure pointer. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvokeDisposeColorPickMethodUPP` (page 1418) Deprecated in Mac OS X v10.4
Invokes a method disposal callback, using a universal procedure pointer. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvokeInitPickMethodUPP` (page 1419) Deprecated in Mac OS X v10.4
Invokes a method initialization callback, using a universal procedure pointer. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvokeRecordColorsUPP` (page 1419) Deprecated in Mac OS X v10.4
Invokes a color recording callback, using a universal procedure pointer. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewCalcColorTableUPP` (page 1420) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a color table calculation callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewDisposeColorPickMethodUPP` (page 1420) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a method disposal callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewInitPickMethodUPP` (page 1421) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a method initialization callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewRecordColorsUPP` (page 1423) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a color recording callback. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Functions

DisposeCalcColorTableUPP

Disposes of a universal procedure pointer (UPP) to a color table calculation callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeCalcColorTableUPP (
   CalcColorTableUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer.

Discussion

For more information, see `CalcColorTableProcPtr` (page 1426).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In

`PictUtils.h`

DisposeDisposeColorPickMethodUPP

Disposes of a universal procedure pointer (UPP) to a method disposal callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeDisposeColorPickMethodUPP (
   DisposeColorPickMethodUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer.

Discussion
For more information, see `DisposeColorPickMethodProcPtr` (page 1428).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

DisposeInitPickMethodUPP

Disposes of a universal procedure pointer (UPP) to a method initialization callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeInitPickMethodUPP (
   InitPickMethodUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer.

Discussion
For more information, see `InitPickMethodProcPtr` (page 1429).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

DisposePictInfo

Disposes of the private data structures allocated by the `NewPictInfo` function. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr DisposePictInfo (
   PictInfoID thePictInfoID
);
```

Parameters

thePictInfoID

The unique identifier returned by `NewPictInfo`.

Return Value

A result code. See "Picture Utilities Result Codes" (page 1441).

Discussion

The `DisposePictInfo` function does not dispose of any of the handles returned to you in a `PictInfo` structure by the `RetrievePictInfo` (page 1425) function. Instead, you can dispose of a `Palette` structure by using the `DisposePalette` function. You can dispose of a `ColorTable` structure by using the `DisposeCTable` function. Dispose of other allocations with the `DisposeHandle` function.

Use this function when you are finished gathering information from a survey of pictures, pixel maps, or bitmaps.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PictUtils.h`

DisposeRecordColorsUPP

Disposes of a universal procedure pointer (UPP) to a color recording callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeRecordColorsUPP (
   RecordColorsUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

For more information, see `RecordColorsProcPtr` (page 1431).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PictUtils.h`

GetPictInfo

Gathers information about a single picture. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr GetPictInfo (
   PicHandle thePictHandle,
   PictInfo *thePictInfo,
   short verb,
   short colorsRequested,
   short colorPickMethod,
   short version
);
```

Parameters

thePictHandle

A handle to a picture.

thePictInfo

On return, a pointer to a `PictInfo` (page 1434) structure, which holds information about the picture. Initially, all of the fields in the new `PictInfo` structure are set to `NULL`. Relevant fields are set to appropriate values depending on the information you request using the `GetPictInfo` function.

This function collects information from black-and-white pictures and bitmaps, and is supported in System 7 even by computers running only basic QuickDraw. However, when collecting color information on a computer running only basic QuickDraw, the function returns `NULL` instead of a handle to a `Palette` or `ColorTable` structure.

verb

A value indicating what type of information you want `GetPictInfo` to return in the `PictInfo` structure. See "Color Information Type" (page 1440) for a description of the values you can use in this parameter.

You can specify whether you want color information (in a `ColorTable` structure, a `Palette` structure, or both), whether you want picture comment information, and whether you want font information. If you want color information, be sure to use the `colorPickMethod` parameter to specify the method by which to select colors.

Because the Palette Manager adds black and white when creating a `Palette` structure, you can specify the number of colors you want minus 2 in the `colorsRequested` parameter and specify the `suppressBlackAndWhite` constant in the `verb` parameter when gathering colors destined for a `Palette` structure or a screen.

colorsRequested

From 1 to 256, the number of colors you want in the `ColorTable` or `Palette` structure returned via the `PictInfo` structure. If you are not requesting colors (that is, if you pass the `recordComments` or `recordFontInfo` constant in the `verb` parameter), specify 0 in this parameter.

colorPickMethod

The method by which colors are selected for the `ColorTable` or `Palette` structure returned via the `PictInfo` structure. See "Color Selection Method" (page 1440) for a description of the values you can use here.

You can also create your own color-picking method in a resource file of type `'cpmt'` and pass its resource ID in the `colorPickMethod` parameter. The resource ID must be greater than 127.

version

Always set this parameter to 0.

Return Value

A result code. See "Picture Utilities Result Codes" (page 1441).

Discussion

The Picture Utilities provide two color-picking methods: one (specified by the `popularMethod` constant) that gives you the most frequently used colors and one (specified by the `medianMethod` constant) that gives you the widest range of colors. Each has advantages in different situations. For example, suppose the picture of a forest image contains 400 colors, of which 300 are greens, 80 are browns, and the rest are a scattering of golden sunlight effects. If you ask for the 250 most used colors, you will probably receive all greens. If you ask for a range of 250 colors, you will receive an assortment stretching from the greens and golds to the browns, including colors in between that might not actually appear in the image. If you specify the `systemMethod` constant, the Picture Utilities choose the method; currently they always choose `popularMethod`. You can also supply a color-picking method of your own.

If your application uses more than one color-picking method, it should present the user with a choice of which method to use.

When you are finished with the information in the `PictInfo` structure, use the Memory Manager function `DisposeHandle` to dispose of the `PictInfo`, `CommentSpec`, and `FontSpec` structures. Dispose of the `Palette` structure by using the `DisposePalette` function. Dispose of the `ColorTable` structure by using the `DisposeCTable` function.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

When you ask for color information, `GetPictInfo` takes into account only the version 2 and extended version 2 picture opcodes `RGBFgCol`, `RGBBkCol`, `BkPixPat`, `PnPixPat`, `FillPixPat`, `HiliteColor` and pixel map or bitmap data. Each occurrence of these opcodes is treated as 1 pixel, regardless of the number and sizes of the objects drawn with that color. If you need an accurate set of colors from a complex picture, create an image of the picture in an offscreen pixel map, and then call the `GetPixMapInfo` (page 1416) function to obtain color information about that pixel map.

The `GetPictInfo` function returns a bit depth of 1 on QuickTime-compressed `'PICT'` files. However, when QuickTime is installed, QuickTime decompresses and displays the image correctly.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PictUtils.h`

GetPixMapInfo

Gathers color information about a single pixel map or bitmap. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr GetPixMapInfo (
   PixMapHandle thePixMapHandle,
   PictInfo *thePictInfo,
   short verb,
   short colorsRequested,
   short colorPickMethod,
   short version
);
```

Parameters

thePixMapHandle

A handle to a pixel map or bitmap.

thePictInfo

On return, a pointer to a `PictInfo` (page 1434) structure, which holds information about a pixel map or bitmap. Initially, all of the fields in a new `PictInfo` structure are set to `NULL`. Relevant fields are set to appropriate values depending on the information you request using the `GetPixMapInfo` function.

This function also collects information from black-and-white pictures and bitmaps, and is supported in System 7 even by computers running only basic QuickDraw. However, when collecting color information on a computer running only basic QuickDraw, this function returns `NULL` instead of a handle to a `Palette` or `ColorTable` structure.

verb

A value indicating whether you want color information returned in a `ColorTable` structure, a `Palette` structure, or both. You can also request that black and white not be included among the returned colors. See "Color Information Type" (page 1440) for a description of the values you can use here.

Because the Palette Manager adds black and white when creating a `Palette` structure, you can specify the number of colors you want minus 2 in the `colorsRequested` parameter and specify the constant `suppressBlackAndWhite` in the `verb` parameter when gathering colors destined for a `Palette` structure or a screen.

colorsRequested

From 1 to 256, the number of colors you want in the `ColorTable` or `Palette` structure returned via the `PictInfo` structure.

colorPickMethod

The method by which colors are selected for the `ColorTable` or `Palette` structure returned via the `PictInfo` structure. See "Color Selection Method" (page 1440) for a description of the values you can use here.

You can also create your own color-picking method in a resource file of type `'cpmt'` and pass its resource ID in the `colorPickMethod` parameter. The resource ID must be greater than 127.

version

Always set this parameter to 0.

Return Value
A result code. See "Picture Utilities Result Codes" (page 1441).

Discussion
The Picture Utilities provide two color-picking methods: one that gives you the most frequently used colors and one that gives you the widest range of colors. If you specify the `systemMethod` constant, the Picture Utilities choose that method. Currently they always choose `popularMethod`. You can also supply a color-picking method of your own.

When you are finished with the information in the `PictInfo` structure, be sure to dispose of it. Use the Memory Manager function `DisposeHandle` to dispose of the `PictInfo` structure. Dispose of the `Palette` structure by using the `DisposePalette` function. Dispose of the `ColorTable` structure by using the `DisposeCTable` function.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PictUtils.h`

InvokeCalcColorTableUPP

Invokes a color table calculation callback, using a universal procedure pointer. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr InvokeCalcColorTableUPP (
   UInt32 dataRef,
   SInt16 colorsRequested,
   void *colorBankPtr,
   CSpecArray resultPtr,
   CalcColorTableUPP userUPP
);
```

Discussion
For parameter descriptions, see `CalcColorTableProcPtr` (page 1426).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

InvokeDisposeColorPickMethodUPP

Invokes a method disposal callback, using a universal procedure pointer. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr InvokeDisposeColorPickMethodUPP (
   UInt32 dataRef,
   DisposeColorPickMethodUPP userUPP
);
```

Discussion
For more information, see `DisposeColorPickMethodProcPtr` (page 1428).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

InvokeInitPickMethodUPP

Invokes a method initialization callback, using a universal procedure pointer. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr InvokeInitPickMethodUPP (
   SInt16 colorsRequested,
   UInt32 *dataRef,
   SInt16 *colorBankType,
   InitPickMethodUPP userUPP
);
```

Discussion
For parameter descriptions, see `InitPickMethodProcPtr` (page 1429).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

InvokeRecordColorsUPP

Invokes a color recording callback, using a universal procedure pointer. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr InvokeRecordColorsUPP (
   UInt32 dataRef,
   RGBColor *colorsArray,
   SInt32 colorCount,
   SInt32 *uniqueColors,
   RecordColorsUPP userUPP
);
```

Discussion
For parameter descriptions, see `RecordColorsProcPtr` (page 1431).

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PictUtils.h`

NewCalcColorTableUPP

Creates a new universal procedure pointer (UPP) to a color table calculation callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CalcColorTableUPP NewCalcColorTableUPP (
   CalcColorTableProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your color table calculation callback. For more information, see `CalcColorTableProcPtr` (page 1426).

Return Value

A UPP to the callback.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PictUtils.h`

NewDisposeColorPickMethodUPP

Creates a new universal procedure pointer (UPP) to a method disposal callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
DisposeColorPickMethodUPP NewDisposeColorPickMethodUPP (
   DisposeColorPickMethodProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your method disposal callback. For more information, see `DisposeColorPickMethodProcPtr` (page 1428).

Return Value
A UPP to the callback.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

NewInitPickMethodUPP

Creates a new universal procedure pointer (UPP) to a method initialization callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
InitPickMethodUPP NewInitPickMethodUPP (
   InitPickMethodProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your method initialization callback. For more information, see `InitPickMethodProcPtr` (page 1429).

Return Value
A UPP to the callback.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

NewPictInfo

Begins collecting pictures, pixel maps, and bitmaps for a survey of pictures. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr NewPictInfo (
   PictInfoID *thePictInfoID,
   short verb,
   short colorsRequested,
   short colorPickMethod,
   short version
);
```

Parameters

thePictInfoID

On return, a value that uniquely identifies your collection of pictures, pixel maps, or bitmaps.

verb

A value indicating what type of information you want the `RetrievePictInfo` (page 1425) function to return in a `PictInfo` (page 1434) structure. See "Color Information Type" (page 1440) for a description of the values you can use here.

The constants `recordComments` and `recordFontInfo` and the values they represent have no effect when gathering information about the pixel maps and bitmaps included in your survey.

Because the Palette Manager adds black and white when creating a palette, you can specify the number of colors you want minus 2 in the `colorsRequested` parameter and specify the constant `suppressBlackAndWhite` in the `verb` parameter when gathering colors destined for a `Palette` structure or a screen.

colorsRequested

From 1 to 256, the number of colors you want included in the `ColorTable` or `Palette` structure returned by the `RetrievePictInfo` function via a `PictInfo` structure.

colorPickMethod

The method by which colors are selected for the `ColorTable` or `Palette` structure included in the `PictInfo` structure returned by the `RetrievePictInfo` function. See "Color Selection Method" (page 1440) for a description of the values you can use here.

You can also create your own color-picking method in a resource file of type `'cpmt'` and pass its resource ID in the `colorPickMethod` parameter. The resource ID must be greater than 127.

version

Always set this parameter to 0.

Return Value

A result code. See "Picture Utilities Result Codes" (page 1441).

Discussion

To add the information for a picture to your survey, use the `RecordPictInfo` function. To add the information for a pixel map or a bitmap to your survey, use the `RecordPixMapInfo` (page 1424) function. For each of these functions, identify the survey with the ID number returned by `NewPictInfo`.

Use the `RetrievePictInfo` function to return information about the pictures, pixel maps, and bitmaps in the survey. The `RetrievePictInfo` function returns your requested information in a `PictInfo` structure.

Use the `verb` parameter for `NewPictInfo` to specify whether you want to gather comment or font information for the pictures in the survey. If you want to gather color information, use the `verb` parameter for `NewPictInfo` to specify whether you want this information in a `ColorTable` structure, a `Palette` structure,

or both. The `PictInfo` structure returned by the `RetrievePictInfo` function will then include a handle to a `ColorTable` structure or a `Palette` structure, or handles to both. If you want color information, be sure to use the `colorPickMethod` parameter to specify the method by which to select colors.

The Picture Utilities provide two color-picking methods: one (specified by the `popularMethod` constant) that gives you the most frequently used colors and one (specified by the `medianMethod` constant) that gives you the widest range of colors. If you specify the `systemMethod` constant, the Picture Utilities choose the method; currently they always choose `popularMethod`. You can also supply a color-picking method of your own.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PictUtils.h`

NewRecordColorsUPP

Creates a new universal procedure pointer (UPP) to a color recording callback. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RecordColorsUPP NewRecordColorsUPP (
   RecordColorsProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your color recording callback. For more information, see `RecordColorsProcPtr` (page 1431).

Return Value
A UPP to the callback.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PictUtils.h`

RecordPictInfo

Adds a picture to an informational survey of multiple pictures. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr RecordPictInfo (
   PictInfoID thePictInfoID,
   PicHandle thePictHandle
);
```

Parameters

thePictInfoID

The ID number—returned by the `NewPictInfo` (page 1422) function—that identifies the survey to which you are adding the picture.

thePictHandle

A handle to the picture being added to the survey.

Return Value

A result code. See "Picture Utilities Result Codes" (page 1441).

Discussion

The `RecordPictInfo` function adds the picture you specify in the parameter `thePictHandle` to the survey of pictures identified by the parameter `thePictInfoID`. Use `RecordPictInfo` repeatedly to add additional pictures to your survey.

After you have collected all of the pictures you need, use the `RetrievePictInfo` (page 1425) function to return information about pictures in the survey.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

When you ask for color information, `RecordPictInfo` takes into account only the version 2 and extended version picture opcodes `RGBFgCol`, `RGBBkCol`, `BkPixPat`, `PnPixPat`, `FillPixPat`, and `HiliteColor`. Each occurrence of these opcodes is treated as 1 pixel, regardless of the number and sizes of the objects drawn with that color. If you need an accurate set of colors from a complex picture, create an image of the picture in an offscreen pixel map, and then call the `GetPixMapInfo` (page 1416) function to obtain color information about that pixel map.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PictUtils.h`

RecordPixMapInfo

Adds a pixel map or bitmap to an informational survey of multiple pixel maps and bitmaps. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr RecordPixMapInfo (
   PictInfoID thePictInfoID,
   PixMapHandle thePixMapHandle
);
```

Parameters

thePictInfoID

The ID number—returned by the `NewPictInfo` (page 1422) function—that identifies the survey to which you are adding the pixel map or bitmap.

thePixMapHandle

A handle to a pixel map or bitmap to be added to the survey.

Return Value

A result code. See "Picture Utilities Result Codes" (page 1441).

Discussion

The `RecordPixMapInfo` function adds the pixel map or bitmap you specify in the parameter `thePixMapHandle` to the survey identified by the parameter `thePictInfoID`. Use `RecordPictInfo` repeatedly to add additional pixel maps and bitmaps to your survey.

After you have collected all of the images you need, use the `RetrievePictInfo` (page 1425) function to return information about all the images in the survey.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PictUtils.h`

RetrievePictInfo

Returns information about all the pictures, pixel maps, and bitmaps included in a survey. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr RetrievePictInfo (
   PictInfoID thePictInfoID,
   PictInfo *thePictInfo,
   short colorsRequested
);
```

Parameters

thePictInfoID

The ID number, returned by the `NewPictInfo` (page 1422) function, that identifies the survey of pictures, pixel maps, and bitmaps.

thePictInfo

On return, a pointer to the `PictInfo` (page 1434) structure that holds information about the pictures or images in the survey.

This function also collects information from black-and-white pictures and bitmaps, and is supported in System 7 even by computers running only basic QuickDraw. However, when collecting color information on a computer running only basic QuickDraw, the function returns `NULL` instead of a handle to a `Palette` or `ColorTable` structure.

colorsRequested

From 1 to 256, the number of colors you want returned in the `ColorTable` or `Palette` structure included in the `PictInfo` structure.

Return Value
A result code. See "Picture Utilities Result Codes" (page 1441).

Discussion
After using the `NewPictInfo` function to create a new survey, and then using `RecordPictInfo` to add pictures to your survey and `RecordPixMapInfo` to add pixel maps and bitmaps to your survey, call `RetrievePictInfo`.

When you are finished with the information in the `PictInfo` structure, dispose of the `Palette` structure by using the `DisposePalette` function. Dispose of the `ColorTable` structure with the `DisposeCTable` function. Dispose of other allocations with the `DisposeHandle` function. Use the `DisposePictInfo` function to dispose of the private data structures created by the `NewPictInfo` function.

Special Considerations

Because Quartz 2D uses an entirely different approach to graphics than used by QuickDraw, there is no one-to-one correlation between QuickDraw and Quartz 2D functions. However, because Quartz offers many new features and improved performance compared to QuickDraw, it is worthwhile making the effort to convert your graphics code to Quartz.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`PictUtils.h`

Callbacks

CalcColorTableProcPtr

Defines a pointer to a color table calculation callback. Your color calculation callback selects as many colors as are requested by your application from the color bank for a picture or pixel map and then fills these colors into an array of `ColorSpec` structures.

```
typedef OSErr (*CalcColorTableProcPtr)
(
    UInt32 dataRef,
    SInt16 colorsRequested,
    void * colorBankPtr,
    CSpecArray resultPtr
);
```

If you name your function `MyCalcColorTableProc`, you would declare it like this:

```
OSErr CalcColorTableProcPtr (
    UInt32 dataRef,
    SInt16 colorsRequested,
    void * colorBankPtr,
    CSpecArray resultPtr
);
```

Parameters

dataRef

A handle to any data your method needs. Your application initially creates this handle using the `InitPickMethodProcPtr` (page 1429) function.

colorsRequested

The number of colors requested by your application to be gathered for examination in a `ColorTable` or `Palette` structure.

colorBankPtr

If your `MyInitPickMethodCallback` function returned either the `colorBankIsExactAnd555` or `colorBankIs555` constant, then this parameter contains a pointer to the 5-5-5 histogram that describes all of the colors in the picture, pixel map, or bitmap being examined. (The format of the 5-5-5 histogram is explained in the function description for the `InitPickMethodProcPtr` (page 1429) function.) Your `MyCalcColorTableCallback` function should examine these colors and then, using its own criterion for selecting the colors, fill in an array of `ColorSpec` structures with the number of colors specified in the `colorsRequested` parameter.

If your `MyInitPickMethodCallback` function returned the `colorBankIsCustom` constant, then the value passed in this parameter is invalid. In this case, your `MyCalcColorTableCallback` function should use the custom color bank that your application created (using the `RecordColorsProcPtr` (page 1431) function) for filling in an array of `ColorSpec` structures with the number of colors specified in the `colorsRequested` parameter.

Your `MyCalcColorTableCallback` function should return a pointer to this array of `ColorSpec` structures in the next parameter.

resultPtr

A pointer to the array of `ColorSpec` structures to be filled with the number of colors specified in the `colorsRequested` parameter. The Picture Utilities function that your application initially called places these colors in a `Palette` structure or `ColorTable` structure, as specified by your application.

Return Value

A result code. See "Picture Utilities Result Codes" (page 1441). If `MyCalcColorTableCallback` generates an error, it should return the error as its function result. This error is passed back to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function, which in turn passes the error to your application as a function result.

Discussion
Selecting from the color bank created for the picture, bitmap, or pixel map being examined, `MyCalcColorTableCallback` fills an array of `ColorSpec` structures with the number of colors requested in the `colorsRequested` parameter and returns this array in the `resultPtr` parameter.

If more colors are requested than the picture contains, `MyCalcColorTable` fills the remaining entries with black (0000 0000 0000).

The `colorBankPtr` parameter is of type `Ptr` because the data stored in the color bank is of the type specified by your `InitPickMethodProcPtr` (page 1429) function. Thus, if you specified `colorBankIs555` in the `colorBankType` parameter, the color bank would be an array of integers. However, if the Picture Utilities support other data types in the future, the `colorBankPtr` parameter could point to completely different data types.

Always coerce the value passed in the `colorBankPtr` parameter to a pointer to an integer. In the future you may need to coerce this value to a pointer of the type you specify in your `MyInitPickMethodCallback` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

DisposeColorPickMethodProcPtr

Defines a pointer to a method disposal callback function. Your method disposal function releases the memory for the `'cpmt'` resource allocated by your `MyInitPickMethodCallback` function.

```
typedef OSErr (*DisposeColorPickMethodProcPtr)
(
    UInt32 dataRef
);
```

If you name your function `MyDisposeColorPickMethodProc`, you would declare it like this:

```
OSErr DisposeColorPickMethodProcPtr (
    UInt32 dataRef
);
```

Parameters
`dataRef`
A handle to any data your method needs. Your application initially creates this handle using the `InitPickMethodProcPtr` (page 1429) function.

Return Value
A result code. See "Picture Utilities Result Codes" (page 1441). If your `MyDisposeColorPickMethodCallback` function generates an error, it should return the error as its function result. This error is passed back to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function, which in turn passes the error to your application as a function result.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

InitPickMethodProcPtr

Defines a pointer to a method initialization callback function. Your method initialization function specifies the color back and allocates whatever data your color-picking method needs.

```
typedef OSErr (*InitPickMethodProcPtr)
(
    SInt16 colorsRequested,
    UInt32 * dataRef,
    SInt16 * colorBankType
);
```

If you name your function `MyInitPickMethodProc`, you would declare it like this:

```
OSErr InitPickMethodProcPtr (
    SInt16 colorsRequested,
    UInt32 * dataRef,
    SInt16 * colorBankType
);
```

Parameters

colorsRequested

The number of colors requested by your application to be gathered for examination in a `ColorTable` or `Palette` structure.

dataRef

A handle to any data needed by your color-picking method; that is, if your application allocates and uses additional data, it should return a handle to it in this parameter.

colorBankType

The type of color bank your color-picking method uses. Your `MyInitPickMethodCallback` function should return one of three valid color bank types.

Return the `colorBankIs555` constant in this parameter if you want to let the Picture Utilities gather the colors for a picture or a pixel map into a 5-5-5 histogram. When you return the `colorBankIs555` constant, the Picture Utilities call your `MyCalcColorTableCallback` function with a pointer to the color bank (that is, to the 5-5-5 histogram). Your `MyCalcColorTableCallback` function selects whatever colors it needs from this color bank. Then the Picture Utilities function called by your application returns these colors in a `Palette` structure, a `ColorTable` structure, or both, as requested by your application.

Return the `ColorBankIsExactAnd555` constant in this parameter to make the Picture Utilities return exact colors if there are less than 256 unique colors in the picture; otherwise, the Picture Utilities gather the colors for the picture in a 5-5-5 histogram, just as they do when you return the `colorBankIs555` constant. If the picture or pixel map has fewer colors than your application requests when it calls a Picture Utilities function, the Picture Utilities function returns all of the colors contained in the color bank. If the picture or pixel map contains more colors than your application requests, the Picture Utilities call your `MyCalcColorTableCallback` function to select which colors to return.

Return the `colorBankIsCustom` constant in this parameter if you want to implement your own color bank for storing the colors in a picture or a pixel map. For example, because the 5-5-5 histogram that the Picture Utilities provide gathers colors to a resolution of 5 bits per color, your application may want to create a histogram with a resolution of 8 bits per color. When you return the `colorBankIsCustom` constant, the Picture Utilities call your `MyRecordColorsCallback` function to create this color bank. The Picture Utilities also call your `MyCalcColorTableCallback` function to select colors from this color bank.

Return Value
A result code. See "Picture Utilities Result Codes" (page 1441). If `MyInitPickMethodCallback` generates any error, it should return the error as its function result. This error is passed back to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function, which in turn passes the error to your application as a function result.

Discussion
Your color-picking method (`'cpmt'`) resource should include a function that specifies its color bank (that is, the structure into which all the colors of a picture, pixel map, or bitmap are gathered) and allocates whatever data your color-picking method needs. Your `MyInitPickMethodCallback` can let the Picture Utilities generate a color bank consisting of a histogram (that is, frequency counts of each color) to a resolution of 5 bits per color. Or, your `MyInitPickMethodCallback` function can specify that your application has its own custom color bank—for example, a histogram to a resolution of 8 bits per color.

The 5-5-5 histogram that the Picture Utilities provide if you return the `ColorBankIs555` or `ColorBankIsExactAnd555` constant in the `colorBankType` parameter is like a reversed `cSpecArray` structure, which is an array of `ColorSpec` structures. This 5-5-5 histogram is an array of 32,768 integers, where the index into the array is the color: 5 bits of red, followed by 5 bits of green, followed by 5 bits of blue. Each entry in the array is the number of colors in the picture that are approximated by the index color for that entry.

For example, suppose there were three instances of the following color in the pixel map:

- Red = %1101 1010 1010 1110
- Green = %0111 1010 1011 0001
- Blue = %0101 1011 0110 1010

This color would be represented by index % 0 11011-01111-01011 (in hexadecimal, $6DEB), and the value in the histogram at this index would be 3, because there are three instances of this color.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

RecordColorsProcPtr

Defines a pointer to a color recording callback function. Your color recording function creates a color bank.

```
typedef OSErr (*RecordColorsProcPtr)
(
    UInt32 dataRef,
    RGBColor * colorsArray,
    SInt32 colorCount,
    SInt32 * uniqueColors
);
```

If you name your function `MyRecordColorsProc`, you would declare it like this:

```
OSErr RecordColorsProcPtr (
    UInt32 dataRef,
    RGBColor * colorsArray,
    SInt32 colorCount,
    SInt32 * uniqueColors
);
```

Parameters

`dataRef`
A handle to any data your function needs. Your application initially creates this handle using the `InitPickMethodProcPtr` (page 1429) function.

`colorsArray`
An array of `RGBColor` structures. Your `MyRecordColorsCallback` function stores the color information for this array of `RGBColor` structures in a data structure of type `RGBColorArray`.

`colorCount`
The number of colors in the array specified in the `colorsArray` parameter.

`uniqueColors`
Upon input, the number of unique colors already added to the array in the `colorsArray` parameter. (The Picture Utilities functions call your `MyRecordColors` function once for every color in the picture, pixel map, or bitmap.) Your `MyRecordColorsCallback` function must calculate the number of unique colors (to the resolution of the color bank) that are added by this call. Your `MyRecordColorsCallback` function should add this amount to the value passed upon input in this parameter and then return the sum in this parameter.

Return Value
A result code. See "Picture Utilities Result Codes" (page 1441). If your `MyRecordColorsCallback` function generates any error, it should return the error as its function result. This error is passed back to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function, which in turn passes the error to your application as a function result.

Discussion
`MyRecordColorsCallback` stores each color encountered in a picture or pixel into its own color bank. The Picture Utilities call `MyRecordColorsCallback` only if your `MyInitPickMethodCallback` function returns the constant `colorBankIsCustom` in the `colorBankType` parameter. When you return the `colorBankIsCustom` constant in the `colorBankType` parameter to your `MyInitPickMethodCallback` function, your color-picking method (`'cpmt'`) resource must include a function that creates this color bank; for example, your application may want to create a histogram with a resolution of 8 bits per color.

The Picture Utilities functions call `MyRecordColorsCallback` for all the colors in the picture, pixel map, or bitmap.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

Data Types

CalcColorTableUPP

Defines a universal procedure pointer (UPP) to a color table calculation callback.

```
typedef CalcColorTableProcPtr CalcColorTableUPP;
```

Discussion
For more information, see the description of the callback function `CalcColorTableProcPtr` (page 1426).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

CommentSpec

Contains information about the comments in a picture.

```
struct CommentSpec {
    short count;
    short ID;
};
typedef struct CommentSpec CommentSpec;
typedef CommentSpec * CommentSpecPtr;
typedef CommentSpecPtr * CommentSpecHandle;
```

Fields
`count`
The number of times this kind of picture comment occurs in the picture specified to the `GetPictInfo` function or in all the pictures examined with the `NewPictInfo` function.

`ID`

The value set in the `kind` parameter when the picture comment was created using the function `PicComment`. For a description of this function, see *Inside Mac OS X: Quickdraw Reference*.

Discussion
If you specify the `structureComments` constant in the `verb` parameter to the `GetPictInfo` (page 1414) function or the `NewPictInfo` (page 1422) function, you receive a `PictInfo` (page 1434) structure that includes in its `commentHandle` field a handle to an array of `CommentSpec` structures. The `uniqueComments` field of the `PictInfo` structure indicates the number of `CommentSpec` structures in this array.

When you are finished using the information returned in a `CommentSpec` structure, use the `DisposeHandle` function to dispose of the memory allocated to it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

DisposeColorPickMethodUPP

Defines a universal procedure pointer (UPP) to a method disposal callback.

```
typedef DisposeColorPickMethodProcPtr DisposeColorPickMethodUPP;
```

Discussion
For more information, see the description of the callback function `DisposeColorPickMethodProcPtr` (page 1428).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

FontSpec

Contains information about the fonts in a picture.

```
struct FontSpec {
    short pictFontID;
    short sysFontID;
    long size[4];
    short style;
    long nameOffset;
};
typedef struct FontSpec FontSpec;
typedef FontSpec * FontSpecPtr;
typedef FontSpecPtr * FontSpecHandle;
```

Fields
`pictFontID`

The ID number of the font as it is stored in the picture.

`sysFontID`
: The number that identifies the resource file (of type `'FOND'`) that specifies the font family. Every font family, has a unique font family ID, in a range of values that determines the script system to which the font family belongs.

`size`
: The point sizes of the fonts in the picture. The field contains 128 bits, in which a bit is set for each point size encountered, from 1 to 127 points. Bit 0 is set if a size larger than 127 is found.

`style`
: The styles for this font family at any of its sizes. The values in this field can also be represented with the `Style` data type.

`nameOffset`
: The offset into the list of font names (indicated by the `fontNamesHandle` field of the `PictInfo` structure) at which the name for this font family is stored. A font name is given to a font family to distinguish it from other font families.

Discussion
If you specify the `recordFontInfo` constant in the `verb` parameter to the `GetPictInfo` function or the `NewPictInfo` function, your application receives a `PictInfo` structure that includes in its `fontHandle` field a handle to an array of `FontSpec` structures. The `uniqueFonts` field of the `PictInfo` structure indicates the number of `FontSpec` structures in this array. (For bitmap fonts, a font is a complete set of glyphs in one size, typeface, and style. For outline fonts, a font is a complete set of glyphs in one typeface and style.)

When you are finished using the information returned in a `FontSpec` structure, you should use the Memory Manager function `DisposeHandle` to dispose of the memory allocated to it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

InitPickMethodUPP

Defines a universal procedure pointer (UPP) to a method initialization callback.

```
typedef InitPickMethodProcPtr InitPickMethodUPP;
```

Discussion
For more information, see the description of the callback function `InitPickMethodProcPtr` (page 1429).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

PictInfo

Contains information about a picture.

```
struct PictInfo {
    short version;
    long uniqueColors;
    PaletteHandle thePalette;
    CTabHandle theColorTable;
    Fixed hRes;
    Fixed vRes;
    short depth;
    Rect sourceRect;
    long textCount;
    long lineCount;
    long rectCount;
    long rRectCount;
    long ovalCount;
    long arcCount;
    long polyCount;
    long regionCount;
    long bitMapCount;
    long pixMapCount;
    long commentCount;
    long uniqueComments;
    CommentSpecHandle commentHandle;
    long uniqueFonts;
    FontSpecHandle fontHandle;
    Handle fontNamesHandle;
    long reserved1;
    long reserved2;
};
typedef struct PictInfo PictInfo;
typedef PictInfo * PictInfoPtr;
```

Fields

`version`

The version number of the Picture Utilities, currently set to 0.

`uniqueColors`

The number of colors in the picture specified to the `GetPictInfo` function, or the number of colors in the pixel map or bitmap specified to the `GetPixMapInfo` function, or the total number of colors for all the pictures, pixel maps, and bitmaps returned by the `RetrievePictInfo` function. The number of colors returned in this field is limited by the accuracy of the Picture Utilities' color bank for color storage. See `InitPickMethodProcPtr` (page 1429), `RecordColorsProcPtr` (page 1431), `CalcColorTableProcPtr` (page 1426), and `DisposeColorPickMethodProcPtr` (page 1428) for information about the Picture Utility's color bank and about how you can create your own for selecting colors.

`thePalette`

A handle to the resulting `Palette` structure if you specified to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function that colors be returned in a `Palette` structure. That `Palette` structure contains either the number of colors you specified to the function or—if there are not that many colors in the pictures, pixel maps, or bitmaps—the number of colors found. Depending on the constant you pass in the `verb` parameter to the function, the `Palette` structure contains either the most used or the widest range of colors in the pictures, pixel maps, and bitmaps. On Macintosh computers running basic QuickDraw only, this field is always returned as `NULL`.

`theColorTable`

A handle to the resulting `ColorTable` structure if you specified to the `GetPictInfo`, `GetPixMapInfo`, or `NewPictInfo` function that colors be returned in a `ColorTable` structure. If the pictures, pixel maps, or bitmaps contain fewer colors found than you specified to the function, the unused entries in the `ColorTable` structure are filled with black. Depending on the constant you pass in the `verb` parameter to the function, the `ColorTable` structure contains either the most used or the widest range of colors in the pictures, pixel maps, and bitmaps. On Macintosh computers running basic QuickDraw only, this field is always returned as `NULL`.

If a picture has more than 256 colors or has pixel depths of 32 bits, then Color QuickDraw translates the colors in the `ColorTable` structure to 16-bit depths. In such a case, the returned colors might have a slight loss of resolution, and the `uniqueColors` field reflects the number of colors distinguishable at that pixel depth.

`hRes`

The horizontal resolution of the current picture, pixel map, or bitmap retrieved by the `GetPictInfo` or `GetPixMapInfo` function or the greatest horizontal resolution from all pictures, pixel maps, and bitmaps retrieved by the `RetrievePictInfo` function.

`vRes`

The vertical resolution of the current picture, pixel map, or bitmap retrieved by the `GetPictInfo` or `GetPixMapInfo` function or the greatest vertical resolution of all pictures, pixel maps, and bitmaps retrieved by the `RetrievePictInfo` function. Although the values of the `hRes` and `vRes` fields are usually the same, they do not have to be.

`depth`

The pixel depth of the picture specified to the `GetPictInfo` function or the pixel map specified to the `GetPixMapInfo` function. When you use the `RetrievePictInfo` function, this field contains the deepest pixel depth of all pictures or pixel maps retrieved by the function.

`sourceRect`

The optimal bounding rectangle for displaying the picture at the resolution indicated by the `hRes` and `vRes` fields. The upper-left corner of the rectangle is always (0,0). Pictures created with the `OpenCPicture` function have the `hRes`, `vRes`, and `sourceRect` fields built into their `Picture` structures. For pictures created by `OpenPicture`, the `hRes` and `vRes` fields are set to 72 dpi, and the source rectangle is calculated using the `picFrame` field of the `Picture` structure for the picture.

`textCount`

The number of text strings in the picture specified to the `GetPictInfo` function, or the total number of text objects in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps specified to `GetPixMapInfo` or `RetrievePictInfo`, this field is set to 0.

`lineCount`

The number of lines in the picture specified to the `GetPictInfo` function, or the total number of lines in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`rectCount`

The number of rectangles in the picture specified to the `GetPictInfo` function, or the total number of rectangles in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`rRectCount`

The number of rounded rectangles in the picture specified to the `GetPictInfo` function, or the total number of rounded rectangles in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`ovalCount`
: The number of ovals in the picture specified to the `GetPictInfo` function, or the total number of ovals in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`arcCount`
: The number of arcs and wedges in the picture specified to the `GetPictInfo` function, or the total number of arcs and wedges in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`polyCount`
: The number of polygons in the picture specified to the `GetPictInfo` function, or the total number of polygons in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`regionCount`
: The number of regions in the picture specified to the `GetPictInfo` function, or the total number of regions in all the pictures retrieved by the `RetrievePictInfo` function. For pixel maps and bitmaps, this field is set to 0.

`bitMapCount`
: The total number of bitmaps in the survey.

`pixMapCount`
: The total number of pixel maps in the survey.

`commentCount`
: The number of comments in the picture specified to the `GetPictInfo` function, or the total number of comments in all the pictures retrieved by the `RetrievePictInfo` function. This field is valid only if you specified to the `GetPictInfo` or `NewPictInfo` function that comments be returned in a `CommentSpec` structure. For pixel maps and bitmaps, this field is set to 0.

`uniqueComments`
: The number of picture comments that have different IDs in the picture specified to the `GetPictInfo` function, or the total number of picture comments with different IDs in all the pictures retrieved by the `RetrievePictInfo` function. This field is valid only if you specify that comments be returned in a `CommentSpec` (page 1432) structure. For pixel maps and bitmaps, this field is set to 0.

`commentHandle`
: A handle to an array of `CommentSpec` structures. For pixel maps and bitmaps, this field is set to `NULL`. See `CommentSpec` (page 1432).

`uniqueFonts`
: The number of different fonts in the picture specified to the `GetPictInfo` function, or the total number of different fonts in all the pictures retrieved by the `RetrievePictInfo` function. For bitmap fonts, a font is a complete set of glyphs in one size, typeface, and style. For outline fonts, a font is a complete set of glyphs in one typeface and style—for example, 12-point Geneva italic. For outline fonts, a font is a complete set of glyphs in one typeface and style—for example, Geneva italic.

  This field is valid only if you specify that fonts be returned in a `FontSpec` (page 1433) structure. For pixel maps and bitmaps, this field is set to 0.

`fontHandle`
: A handle to a list of `FontSpec` structures. For pixel maps and bitmaps, this field is set to `NULL`.

`fontNamesHandle`
: A handle to the names of the fonts in the picture retrieved by the `GetPictInfo` function or the pictures retrieved by the `RetrievePictInfo` function. The offset to a particular name is stored in the `nameOffset` field of the `FontSpec` structure for that font. A font name is a name given to one font family to distinguish it from other font families.

```
reserved1
reserved2
```

Discussion
When you use the `GetPictInfo` (page 1414) function to collect information about a picture, or when you use the `GetPixMapInfo` (page 1416) function to collect color information about a pixel map or bitmap, the function returns the information in a `PictInfo` structure. When you gather this information for multiple pictures, pixel maps, or bitmaps, the `RetrievePictInfo` (page 1425) function also returns a `PictInfo` structure containing this information.

Initially, all of the fields in a new `PictInfo` structure are set to `NULL`. Relevant fields are set to appropriate values depending on the information you request using the Picture Utilities functions.

When you are finished with this information, be sure to dispose of it. You can dispose of `Palette` structures by using the Palette Manager function, `DisposePalette`. Dispose of `ColorTable` structures by using the QuickDraw function, `DisposeCTable`. Dispose of other allocations with the Memory Manager function, `DisposeHandle`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

PictInfoID

Defines an identifier for a collection of pictures, pixel maps, or bitmaps in an application.

```
typedef long PictInfoID;
```

Discussion
Picture Utilities returns a `PictInfoID` value when you call the function `NewPictInfo` (page 1422). It serves as a unique identifier for a collection of pictures, pixel maps, or bitmaps defined in your application. You use this ID when calling other Picture Utilities functions to manage and survey your collection.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PictUtils.h`

RecordColorsUPP

Defines a universal procedure pointer (UPP) to a color recording callback.

```
typedef RecordColorsProcPtr RecordColorsUPP;
```

Discussion
For more information, see the description of the callback function `RecordColorsProcPtr` (page 1431).

Availability
Available in Mac OS X v10.0 and later.

Declared In
PictUtils.h

Constants

Color Bank Type

Specifies the type of color bank used in a color-picking method.

```
enum {
    ColorBankIsCustom = -1,
    ColorBankIsExactAnd555 = 0,
    ColorBankIs555 = 1
};
```

Constants

ColorBankIsCustom
: Gathers colors into a custom color bank. Picture Utilities gathers the colors for a picture or a pixel map into a 5-5-5 histogram. When you return the colorBankIs555 constant, the Picture Utilities call your RecordColorsProcPtr (page 1431) function with a pointer to the color bank (that is, to the 5-5-5 histogram). Your CalcColorTableProcPtr function selects whatever colors it needs from this color bank. Then the Picture Utilities function called by your application returns these colors in a Palette structure, a ColorTable structure, or both, as requested by your application.

Available in Mac OS X v10.0 and later.

Declared in PictUtils.h.

ColorBankIsExactAnd555
: Gathers exact colors if there are less than 256 unique colors in picture; otherwise gathers colors for picture in a 5-5-5 histogram. If the picture or pixel map has fewer colors than your application requests when it calls a Picture Utilities function, the Picture Utilities function returns all of the colors contained in the color bank. If the picture or pixel map contains more colors than your application requests, the Picture Utilities call your CalcColorTableProcPtr function to select which colors to return.

Available in Mac OS X v10.0 and later.

Declared in PictUtils.h.

ColorBankIs555
: Gathers colors into a 5-5-5 histogram. Specify colorBankIsCustom constant if you want to implement your own color bank for storing the colors in a picture or a pixel map. For example, because the 5-5-5 histogram that the Picture Utilities provide gathers colors to a resolution of 5 bits per color, your application may want to create a histogram with a resolution of 8 bits per color. When you return the colorBankIsCustom constant, the Picture Utilities call your RecordColorsProcPtr (page 1431) function to create this color bank. The Picture Utilities also call your CalcColorTableProcPtr function to select colors from this color bank.

Available in Mac OS X v10.0 and later.

Declared in PictUtils.h.

Discussion
Your InitPickMethodProcPtr (page 1429) function returns these constants in the colorBankType parameter to indicate the type of color bank used in your color-picking method.

Color Selection Method

Indicates the color selection method used in a `PictInfo` record.

```
enum {
    systemMethod = 0,
    popularMethod = 1,
    medianMethod = 2
};
```

Constants

`systemMethod`
Lets Picture Utilities choose the method. Currently they always choose `popularMethod`.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

`popularMethod`
Returns the most frequently used colors.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

`medianMethod`
Returns a weighted distribution of colors.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

Discussion

These constants are used to indicate the method by which colors are selected for the `ColorTable` or `Palette` structure returned via the `PictInfo` structure, by the functions `NewPictInfo` (page 1422) , `GetPixMapInfo` (page 1416) , or `GetPictInfo` (page 1414).

Color Information Type

Indicates the type of color information returned in a `PictInfo` record.

```
enum {
    returnColorTable = 0x0001,
    returnPalette = 0x0002,
    recordComments = 0x0004,
    recordFontInfo = 0x0008,
    suppressBlackAndWhite = 0x0010
};
```

Constants

`returnColorTable`
Specify to return a Color Table.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

`returnPalette`
Specify to return a Palette structure.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

`recordComments`
Specify to return comment information.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

`recordFontInfo`
Specify to return font information.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

`suppressBlackAndWhite`
Don't include black and white with returned colors.
Available in Mac OS X v10.0 and later.
Declared in `PictUtils.h`.

Discussion
These constants are used in the `verb` parameter of the `GetPictInfo` (page 1414) , `GetPixMapInfo` (page 1416) ,and `NewPictInfo` (page 1422) functions to indicate the type of information those functions should return. You can use any or all of these constants or the sum of the integers they represent.

Result Codes

The table below lists the most common result codes returned by Picture Utilities.

Result Code	Value	Description
`pictInfoVersionErr`	-11000	Wrong version of the PictInfo structure. Available in Mac OS X v10.0 and later.
`pictInfoIDErr`	-11001	The internal consistancy check for the PictInfoID is wrong. Available in Mac OS X v10.0 and later.
`pictInfoVerbErr`	-11002	The PictInfo verb is not valid. Available in Mac OS X v10.0 and later.
`cantLoadPickMethodErr`	-11003	Unable to load the custom pick method resource. Available in Mac OS X v10.0 and later.
`colorsRequestedErr`	-11004	The number of colors requested is illegal. Available in Mac OS X v10.0 and later.
`pictureDataErr`	-11005	The picture data is not valid. Available in Mac OS X v10.0 and later.

Process Manager Reference

Framework: Carbon/Carbon.h
Declared in Processes.h

Overview

The Process Manager provides the cooperative multitasking environment for versions of Mac OS that preceded Mac OS X. The Process Manager controls access to shared resources and manages the scheduling and execution of applications.

You can use the Process Manager to control the execution of processes and to get information about processes, including your own. You can use the Process Manager routines to

- control the execution of your application
- get information about processes
- launch other applications

Some Process Manager functions access a `ProcessInfoRec` data structure, which contains fields that are no longer applicable in a preemptively scheduled environment (for example, the `processLocation`, `processFreeMem`, and `processActiveTime` fields). Your application should avoid accessing such fields. Changes to the memory model may also affect certain fields.

Carbon does not support Process Manager functions that deal with control panels or desk accessories.

Functions by Task

Getting Process Information

`CopyProcessName` (page 1445)
Gets a copy of the name of a process.

`GetCurrentProcess` (page 1446)
Gets information about the current process, if any.

`GetFrontProcess` (page 1446)
Gets the process serial number of the front process.

`GetNextProcess` (page 1447)
Gets information about the next process, if any, in the Process Manager's internal list of open processes.

`GetProcessBundleLocation` (page 1447)
Retrieves the file system location of the application bundle (or executable file) associated with a process.

`GetProcessInformation` (page 1448)
Get information about a specific process.

`ProcessInformationCopyDictionary` (page 1452)
Obtains a superset of `GetProcessInformation` in modern data types.

`GetProcessPID` (page 1449)
Obtains the Unix PID from a process serial number.

`GetProcessForPID` (page 1448)
Obtains the process serial number from a Unix PID.

`IsProcessVisible` (page 1450)
Determines the visiblility of the user interface for a process.

`SameProcess` (page 1453)
Determines whether two process serial numbers specify the same process.

Starting and Terminating Processes

`LaunchApplication` (page 1451)
Launches an application.

`ExitToShell` (page 1445)
Terminates an application.

`KillProcess` (page 1450)
Terminates a process with the specified ID.

Modifying Processes

`SetFrontProcess` (page 1454)
Moves a process to the foreground.

`SetFrontProcessWithOptions` (page 1455)
Brings a process to the front of the process list, and activates it.

`ShowHideProcess` (page 1455)
Shows or hides a given process.

`TransformProcessType` (page 1456)
Changes the type of the specified process.

`WakeUpProcess` (page 1456)
Makes a suspended process eligible for CPU time.

Functions

CopyProcessName

Gets a copy of the name of a process.

```
OSStatus CopyProcessName (
   const ProcessSerialNumber *psn,
   CFStringRef *name
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 1461) for more information.

name

A Core Foundation string that contains the name of the specified process.

Return Value

A result code. See "Process Manager Result Codes" (page 1467).

Discussion

Because the string returned is a Core Foundation string, it can represent a multilingual name, unlike the `processName` field value you obtain using `GetProcessInformation` (page 1448).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Processes.h`

ExitToShell

Terminates an application.

```
void ExitToShell (
   void
);
```

Discussion

In general, you need to call `ExitToShell` only if you want your application to terminate without reaching the end of its main function.

The `ExitToShell` function terminates the calling process. The Process Manager removes your application from the list of open processes and performs any other necessary cleanup operations. In particular, all memory in your application partition and any temporary memory still allocated to your application is released. If necessary, the Application Died Apple event is sent to the process that launched your application.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

HideMenuBar

ictbSample
QTCarbonShell
QTMetaData

Declared In
`Processes.h`

GetCurrentProcess

Gets information about the current process, if any.

```
OSErr GetCurrentProcess (
    ProcessSerialNumber * PSN
);
```

Parameters

PSN

On output, a pointer to the process serial number of the current process, that is, the one currently accessing the CPU. This application can be running in either the foreground or the background.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
Applications can use this function to find their own process serial number. Drivers can use this function to find the process serial number of the current process. You can use the returned process serial number in other Process Manager functions.

This function is named `MacGetCurrentProcess` on non Macintosh platforms and `GetCurrentProcess` on Macintosh computers. However, even Macintosh code can use the `MacGetCurrentProcess` name because a macro exists that automatically maps that call to `GetCurrentProcess`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetFrontProcess

Gets the process serial number of the front process.

```
OSErr GetFrontProcess (
   ProcessSerialNumber *PSN
);
```

Parameters

PSN

On return, a pointer to the process serial number of the process running in the foreground.

Return Value
A result code. See "Process Manager Result Codes" (page 1467). If no process is running in the foreground, returns `procNotFound`.

Discussion
You can use this function to determine if your process or some other process is in the foreground. You can use the process serial number returned in the `PSN` parameter in other Process Manager functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetNextProcess

Gets information about the next process, if any, in the Process Manager's internal list of open processes.

```
OSErr GetNextProcess (
   ProcessSerialNumber *PSN
);
```

Parameters

PSN

On input, a pointer to the process serial number of a process. This number should be a valid process serial number returned from `LaunchApplication` (page 1451), `GetFrontProcess` (page 1446), or `GetCurrentProcess` (page 1446), or a process serial number structure containing `kNoProcess`. For details about this structure, see `ProcessSerialNumber` (page 1461). On return, a pointer to the process serial number of the next process, or else `kNoProcess`.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
The Process Manager maintains a list of all open processes. You can derive this list by using repetitive calls to `GetNextProcess`. Begin generating the list by calling `GetNextProcess` and specifying the constant `kNoProcess` in the `PSN` parameter. You can then use the returned process serial number to get the process serial number of the next process. Note that the order of the list of processes is internal to the Process Manager. When the end of the list is reached, `GetNextProcess` returns the constant `kNoProcess` in the `PSN` parameter and the result code `procNotFound`.

You can use the returned process serial number in other Process Manager functions. You can also use this process serial number to specify a target application when your application sends a high-level event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetProcessBundleLocation

Retrieves the file system location of the application bundle (or executable file) associated with a process.

```
OSStatus GetProcessBundleLocation (
   const ProcessSerialNumber *psn,
   FSRef *location
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 1461) for more information.

location

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`Processes.h`

GetProcessForPID

Obtains the process serial number from a Unix PID.

```
OSStatus GetProcessForPID (
   pid_t pid,
   ProcessSerialNumber *psn
);
```

Parameters

pid

The Unix process ID (PID).

psn

On return, `psn` points to the process serial number.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
Note that this call does not make sense for Classic applications, since they all share a single UNIX process ID.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetProcessInformation

Get information about a specific process.

```
OSErr GetProcessInformation (
   const ProcessSerialNumber *PSN,
   ProcessInfoRec *info
);
```

Parameters

PSN

A pointer to a valid process serial number. You can pass a process serial number structure containing the constant `kCurrentProcess` to get information about the current process. See `ProcessSerialNumber` (page 1461) for more information.

info

On return, a pointer to a structure containing information about the specified process.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
The information returned in the `info` parameter includes the application's name as it appears in the Application menu, the type and signature of the application, the address of the application partition, the number of bytes in the application partition, the number of free bytes in the application heap, the application that launched the application, the time at which the application was launched, and the location of the application file.

The `GetProcessInformation` function also returns information about the application's `'SIZE'` resource and indicates whether the process is an application or a desk accessory.

You need to specify values for the `processInfoLength`, `processName`, and `processAppSpec` fields of the process information structure. Specify the length of the process information structure in the `processInfoLength` field. If you do not want information returned in the `processName` and `processAppSpec` fields, specify `NULL` for these fields. Otherwise, allocate at least 32 bytes of storage for the string pointed to by the `processName` field and, in the `processAppSpec` field, specify a pointer to an `FSSpec` structure.

The `processName` field may not be what you expect, especially if an application has a localized name. The `processName` field, if not `NULL`, on return will contain the filename part of the executable file of the application. If you want the localized, user-displayable name for an application, call `CopyProcessName` (page 1445).

In Mac OS X, the `processActiveTime` field of the returned structure is always 0, and the `modeCanBackground`, `mode32BitCompatible`, and `modeHighLevelEventAware` fields are always set.

Special Considerations

In most cases, Mac OS X applications should use `ProcessInformationCopyDictionary` (page 1452) instead of this function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

GetProcessPID

Obtains the Unix PID from a process serial number.

```
OSStatus GetProcessPID (
   const ProcessSerialNumber *psn,
   pid_t *pid
);
```

Parameters

psn

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 1461) for more information.

pid

On return, `pid` points to a Unix PID.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
Note that this call does not make sense for Classic applications, since they all share a single UNIX process ID.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

IsProcessVisible

Determines the visiblility of the user interface for a process.

```
Boolean IsProcessVisible (
   const ProcessSerialNumber *psn
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 1461) for more information.

Return Value
Returns `true` if the user interface is currently visible. Otherwise, returns `false`.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Processes.h`

KillProcess

Terminates a process with the specified ID.

```
OSErr KillProcess (
   const ProcessSerialNumber *inProcess
);
```

Parameters

inProcess

The serial number of the process you want to terminate. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 1461) for more information.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
`KillProcess` terminates an process without sending a "quit" Apple event or allowing it any time to save user data or perform cleanup. You should use this function only as a last resort when all other attempts have failed. Even then, there is no guarantee that this call will succeed in killing the application, even if it returns with `noErr`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Processes.h`

LaunchApplication

Launches an application.

```
OSErr LaunchApplication (
   LaunchPBPtr LaunchParams
);
```

Parameters

LaunchParams

A pointer to a `LaunchParamBlockRec` (page 1457) specifying information about the application to launch.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
The `LaunchApplication` function launches the application from the specified file and returns the process serial number, preferred partition size, and minimum partition size if the application is successfully launched.

Note that if you launch another application without terminating your application, the launched application is not actually executed until you make a subsequent call to `WaitNextEvent` or `EventAvail`.

Set the `launchContinue` flag in the `launchControlFlags` field of the launch parameter block if you want your application to continue after the specified application is launched. If you do not set this flag, `LaunchApplication` terminates your application after launching the specified application, even if the launch fails.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

ProcessInformationCopyDictionary

Obtains a superset of `GetProcessInformation` in modern data types.

```
CFDictionaryRef ProcessInformationCopyDictionary (
   const ProcessSerialNumber *PSN,
   UInt32 infoToReturn
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 1461) for more information.

infoToReturn

A bitmask indicating the information to obtain. Pass `kProcessDictionaryIncludeAllInformationMask` for this parameter.

Return Value

An immutable Core Foundation dictionary containing the system information in key-value pairs.

Discussion

You should use this function instead of `GetProcessInformation` (page 1448). Table 41-1 and Table 41-2 show keys you can use to get process attributes in the returned Core Foundation dictionary. All keys in the dictionary are Core Foundation strings. (Note that additional keys exist, but these are for internal use only.) Keys marked with an asterisk (*) may not appear in the dictionary, depending on the application.

Table 41-1 Process information keys

Key	Type	Summary
PSN	`CFNumberRef`	The process serial number. See `ProcessSerialNumber` (page 1461).
Flavor	`CFNumberRef`	A hint as to the type of the application. You shouldn't need to use this key.
Attributes	`CFNumberRef`	Attributes for the process. Useful attributes generally have their own keys.
ParentPSN *	`CFNumberRef`	The process serial number of the application that launched this process.
FileType *	`CFStringRef`	The file type (if any) of the executable.
FileCreator *	`CFStringRef`	The creator type (if any) of the executable.
pid *	`CFNumberRef`	The UNIX PID for this process.
LSBackgroundOnly	`CFBooleanRef`	`kCFBooleanTrue` if the application is a background-only application.

Key	Type	Summary
LSUIElement	`CFBooleanRef`	`kCFBooleanTrue` if the application is an accessibility UIElement.
IsHiddenAttr	`CFBooleanRef`	`kCFBooleanTrue` if the application is currently hidden.
RequiresCarbon	`CFBooleanRef`	`kCFBooleanTrue` if the application's Info.plist file indicates that it is a Carbon application.
LSUIPresentationMode	`CFNumberRef`	The initial user-interface mode for the application. See *Runtime Configuration Guidelines* for a list of possible values.
BundlePath *	`CFStringRef`	The path to the application bundle (if the application is bundled).

Table 41-2 lists additional keys that you should reference by their predefined constants, rather than the actual string names.

Table 41-2 Process information key constants

Key	Type	Summary
`kCFBundleExecutableKey` *	`CFStringRef`	The path to the actual executable file.
`kCFBundleNameKey` *	`CFStringRef`	The application's display name.
`kCFBundleIdentifierKey` *	`CFStringRef`	The application's bundle identifier (if the application is bundled). For example, "com.apple.TextEdit".

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Processes.h`

SameProcess

Determines whether two process serial numbers specify the same process.

```
OSErr SameProcess (
   const ProcessSerialNumber *PSN1,
   const ProcessSerialNumber *PSN2,
   Boolean *result
);
```

Parameters

PSN1

A process serial number.

PSN2

A process serial number.

result

On return, a pointer to a Boolean value which is `TRUE` if the process serial numbers passed in `PSN1` and `PSN2` refer to the same process; otherwise `FALSE`.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
Do not attempt to compare two process serial numbers by any means other than the `SameProcess` function, because the interpretation of the bits in a process serial number is internal to the Process Manager.

The values of `PSN1` and `PSN2` must be valid process serial numbers returned from `LaunchApplication` (page 1451), `GetNextProcess` (page 1447), `GetFrontProcess` (page 1446), `GetCurrentProcess` (page 1446), or a high-level event. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

SetFrontProcess

Moves a process to the foreground.

```
OSErr SetFrontProcess (
   const ProcessSerialNumber *PSN
);
```

Parameters

PSN

A pointer to a valid process serial number. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 1461) for more information.

Return Value
A result code. See "Process Manager Result Codes" (page 1467).

Discussion
The `SetFrontProcess` function moves the specified process to the foreground immediately.

If the specified process serial number is invalid or if the specified process is a background-only application, `SetFrontProcess` returns a nonzero result code and does not change the current foreground process.

Special Considerations

Do not call `SetFrontProcess` at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

SetFrontProcessWithOptions

Brings a process to the front of the process list, and activates it.

```
OSStatus SetFrontProcessWithOptions (
   const ProcessSerialNumber *inProcess,
   OptionBits inOptions
);
```

Parameters

PSN

A pointer to a valid process serial number. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 1461) for more information.

inOptions

A flag that indicates how process windows should be brought forward—see the discussion below.

Return Value

A result code. See "Process Manager Result Codes" (page 1467).

Discussion

If you pass 0 in the `inOptions` parameter, the process is activated and all process windows are brought forward. This is equivalent to calling `SetFrontProcess` (page 1454). If you pass `kSetFrontProcessFrontWindowOnly`, the process is activated and the frontmost nonfloating window is brought forward.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Processes.h`

ShowHideProcess

Shows or hides a given process.

```
OSErr ShowHideProcess (
   const ProcessSerialNumber *psn,
   Boolean visible
);
```

Parameters

PSN

A pointer to a valid process serial number. See `ProcessSerialNumber` (page 1461) for more information.

visible

A Boolean value that specifies whether you want to show (`true`) or hide (`false`) the process.

Return Value

A result code. See "Process Manager Result Codes" (page 1467).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Processes.h`

TransformProcessType

Changes the type of the specified process.

```
OSStatus TransformProcessType (
   const ProcessSerialNumber *psn,
   ProcessApplicationTransformState transformState
);
```

Parameters

PSN

The serial number of the process you want to transform. You can also use the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 1461) for more information.

transformState

A constant indicating the type of transformation you want. See "Process Transformation Constant" (page 1467). Currently you can pass only `kProcessTransformToForegroundApplication`.

Return Value

A result code. See "Process Manager Result Codes" (page 1467).

Discussion

You can use this call to transform a background-only application into a foreground application. A foreground application appears in the Dock (and in the Force Quit dialog) and contains a menu bar. This function does not cause the application to be brought to the front; you must call `SetFrontProcess` (page 1454) to do so.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Processes.h`

WakeUpProcess

Makes a suspended process eligible for CPU time.

```
OSErr WakeUpProcess (
   const ProcessSerialNumber *PSN
);
```

Parameters

PSN

The serial number of the process you want to wake up. You can also pass a process serial number structure containing the constant `kCurrentProcess` to refer to the current process. See `ProcessSerialNumber` (page 1461) for more information.

Return Value

A result code. See "Process Manager Result Codes" (page 1467).

Discussion

The `WakeUpProcess` function makes a process suspended by `WaitNextEvent` eligible to receive CPU time. A process is suspended when the value of the `sleep` parameter in the `WaitNextEvent` function is not 0 and no events for that process are pending in the event queue. This process remains suspended until the time specified in the sleep parameter expires or an event becomes available for that process. You can use

`WakeUpProcess` to make the process eligible for execution before the time specified in the `sleep` parameter expires. This function does not change the order of the processes scheduled for execution; it only makes the specified process eligible for execution.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

Data Types

AppParameters

Defines the first high-level event sent to a newly-launched application.

```
struct AppParameters {
    struct {
        UInt16 what;
        UInt32 message;
        UInt32 when;
        Point where;
        UInt16 modifiers;
    } theMsgEvent;
    unsigned long eventRefCon
    unsigned long messageLength
};
typedef struct AppParameters AppParameters;
typedef AppParameters * AppParametersPtr;
```

Discussion
The application parameters structure is used in the `launchAppParameters` field of the launch parameter block, `LaunchParamBlockRec` (page 1457) , whose address is passed to the `LaunchApplication` (page 1451) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

LaunchParamBlockRec

Defines the required parameters when launching an application.

```
struct LaunchParamBlockRec {
    unsigned long reserved1;
    unsigned short reserved2;
    unsigned short launchBlockID;
    unsigned long launchEPBLength;
    unsigned short launchFileFlags;
    LaunchFlags launchControlFlags;
    FSSpecPtr launchAppSpec;
    ProcessSerialNumber launchProcessSN;
    unsigned long launchPreferredSize;
    unsigned long launchMinimumSize;
    unsigned long launchAvailableSize;
    AppParametersPtr launchAppParameters;
};
typedef struct LaunchParamBlockRec LaunchParamBlockRec;
typedef LaunchParamBlockRec * LaunchPBPtr;
```

Fields

`reserved1`
Reserved.

`reserved2`
Reserved.

`launchBlockID`
A value that indicates whether you are using the fields following it in the launch parameter block. Specify the constant `extendedBlock` if you use the fields that follow it.

`launchEPBLength`
The length of the fields following this field in the launch parameter block. Use the constant `extendedBlockLen` to specify this value.

`launchFileFlags`
The Finder flags for the application file. Set the `launchNoFileFlags` constant in the `launchControlFlags` field if you want the `LaunchApplication` function to extract the Finder flags from the application file and to set the `launchFileFlags` field for you.

`launchControlFlags`
See "Launch Options" (page 1464) for a complete description of these flags.

`launchAppSpec`
A pointer to a file specification structure that gives the location of the application file to launch.

`launchProcessSN`
The process serial number returned to your application if the launch is successful. You can use this process serial number in other Process Manager functions to refer to the launched application.

`launchPreferredSize`
The preferred partition size for the launched application as specified in the launched application's `'SIZE'` resource. `LaunchApplication` sets this field to 0 if an error occurred or if the application is already open.

`launchMinimumSize`
The minimum partition size for the launched application as specified in the launched application's `'SIZE'` resource. `LaunchApplication` sets this field to 0 if an error occurred or if the application is already open.

`launchAvailableSize`
: The maximum partition size that is available for allocation. This value is returned to your application only if the `memFullErr` result code is returned. If the application launch fails because of insufficient memory, you can use this value to determine if there is enough memory available to launch in the minimum size.

`launchAppParameters`
: The first high-level event to send to the launched application. If you set this field to `NULL`, `LaunchApplication` creates and sends the Open Application Apple event to the launched application.

Discussion
You specify a launch parameter block as a parameter to the `LaunchApplication` (page 1451) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

ProcessInfoRec

Defines the structure of a process information record.

```
struct ProcessInfoRec {
    unsigned long processInfoLength;
    StringPtr processName;
    ProcessSerialNumber processNumber;
    unsigned long processType;
    OSType processSignature;
    unsigned long processMode;
    Ptr processLocation;
    unsigned long processSize;
    unsigned long processFreeMem;
    ProcessSerialNumber processLauncher;
    unsigned long processLaunchDate;
    unsigned long processActiveTime;
    FSSpecPtr processAppSpec;
};
typedef struct ProcessInfoRec ProcessInfoRec;
typedef ProcessInfoRec * ProcessInfoRecPtr;
```

Fields

`processInfoLength`
: The number of bytes in the process information structure. For compatibility, you should specify the length of the structure in this field.

`processName`
: The name of the application. This field contains the name of the application as designated by the user at the time the application was opened. For example, for foreground applications, the `processName` field contains the name as it appears in the Application menu. You must specify `NULL` in the `processName` field if you do not want the application name returned. Otherwise, you should allocate at least 32 bytes of storage for the string pointed to by the `processName` field. Note that the `processName` field specifies the name of the application, whereas the `processAppSpec` field specifies the location of the file.

`processNumber`
: The process serial number.

`processType`
: The file type of the application, generally `'APPL'` for applications and `'appe'` for background-only applications launched at startup.

`processSignature`
: The signature (or creator) of the file containing the application.

`processMode`
: Process mode flags. These flags indicate whether the process is an application or desk accessory. For applications, this field also returns information specified in the application's `'SIZE'` resource. This information is returned as flags.

  On Mac OS X, some flags in `processMode` will not be set as they were on Mac OS 9, even for Classic applications. Mac OS X doesn't support applications which can't be sent into the background, so `modeCanBackground` will always be set. Similarly, Mac OS X applications will always have `mode32BitCompatible` and `modeHighLevelEventAware` set

`processLocation`
: The beginning address of the application partition.

`processSize`
: The number of bytes in the application partition (including the heap, stack, and A5 world).

`processFreeMem`
: The number of free bytes in the application heap.

`processLauncher`
: The process serial number of the process that launched the application or desk accessory. If the original launcher of the process is no longer open, the `lowLongOfPSN` field of the process serial number structure contains the constant `kNoProcess`.

`processLaunchDate`
: The value of the `Ticks` global variable at the time that the process was launched.

`processActiveTime`
: The accumulated time, in ticks, during which the process has used the CPU, including both foreground and background processing time.

`processAppSpec`
: The address of a file specification structure that stores the location of the file containing the application or `'DRVR'` resource. You should specify `NULL` in the `processAppSpec` field if you do not want the `FSSpec` structure of the file returned.

Discussion
A process information record is returned by the `GetProcessInformation` (page 1448) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

ProcessInfoExtendedRec

Defines an extended version of the process information record.

```
struct ProcessInfoExtendedRec {
    unsigned long processInfoLength;
    StringPtr processName;
    ProcessSerialNumber processNumber;
    unsigned long processType;
    OSType processSignature;
    unsigned long processMode;
    Ptr processLocation;
    unsigned long processSize;
    unsigned long processFreeMem;
    ProcessSerialNumber processLauncher;
    unsigned long processLaunchDate;
    unsigned long processActiveTime;
    FSSpecPtr processAppSpec;
    unsigned long processTempMemTotal;
    unsigned long processPurgeableTempMemTotal;
};
typedef struct ProcessInfoExtendedRec ProcessInfoExtendedRec;
typedef ProcessInfoExtendedRec * ProcessInfoExtendedRecPtr;
```

Discussion
See `ProcessInfoRec` (page 1459) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

ProcessSerialNumber

Defines the unique identifier for an open process.

```
struct ProcessSerialNumber {
    unsigned long highLongOfPSN;
    unsigned long lowLongOfPSN;
};
typedef struct ProcessSerialNumber ProcessSerialNumber;
typedef ProcessSerialNumber * ProcessSerialNumberPtr;
```

Fields

`highLongOfPSN`
The high-order long integer of the process serial number.

`lowLongOfPSN`
The low-order long integer of the process serial number.

Discussion
All applications (defined as things which can appear in the Dock that are not documents and are launched by the Finder or Dock) on Mac OS X have a unique process serial number. This number is created when the application launches, and remains until the application quits. Other system services, like Apple events, use the `ProcessSerialNumber` structure to specify an application.

During launch, every application "checks in" with the Process Manager. Before this checkin, the application can not receive events or draw to the screen. Prior to Mac OS 10.2, this check in occurred before the applications's `main` function was entered. In Mac OS 10.2 and later, this check in does not occur until the first time the application calls a Process Manager function, or until it enters `CFRunLoopRun` for the main

event loop. This allows tools and other executables which do not need to receive events to link against more of the higher level toolbox frameworks, but may cause a problem if the application expects to be able to retrieve events or use CoreGraphics services before this checkin has occurred. An application can force the connection to the Process Manager to be set up by calling any Process Manager routine, but the recommended way to do this is to call `GetCurrentProcess` (page 1446) to ask for the current application's PSN. Doing so initializes the connection to the Process Manager if it has not already been set up and "check in" the application with the system.

You should not make any assumptions about the meaning of the bits in a process serial number. To compare two process serial numbers, you should use the function `SameProcess` (page 1453).

You can obtain a process serial number in one of the following ways:

- Process serial numbers are returned by the functions `LaunchApplication` (page 1451), `GetCurrentProcess` (page 1446), and `GetFrontProcess` (page 1446).
- Some high-level events return process serial numbers.

If you want to specify a process using the "Process Identification Constants" (page 1466), you must populate a process serial number structure, passing 0 in `highLongOfPSN` and the appropriate constant (such as `kCurrentProcess`) in `lowLongOfPSN`. For example, to bring the current process forward, you can use the following code:

```
ProcessSerialNumber psn = { 0, kCurrentProcess };
SetFrontProcess( &psn );
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTypes.h`

SizeResourceRec

Defines a representation of the `SIZE` resource.

```
struct SizeResourceRec {
    unsigned short flags;
    unsigned long preferredHeapSize;
    unsigned long minimumHeapSize;
};
typedef struct SizeResourceRec SizeResourceRec;
typedef SizeResourceRec * SizeResourceRecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Processes.h`

Constants

Control Panel Result Codes

Specifies the values that a control panel can return.

Unsupported

```
enum {
    cdevGenErr = -1,
    cdevMemErr = 0,
    cdevResErr = 1,
    cdevUnset = 3
};
```

Extension Launch Codes

Specifies the values used when launching extensions.

```
enum {
    extendedBlock = 0x4C43,
    extendedBlockLen = sizeof(LaunchParamBlockRec) - 12
};
```

Control Panel Message Codes

Specifies the values for messages to a control panel.

```
enum {
    initDev = 0,
    hitDev = 1,
    closeDev = 2,
    nulDev = 3,
    updateDev = 4,
    activDev = 5,
    deactivDev = 6,
    keyEvtDev = 7,
    macDev = 8,
    undoDev = 9,
    cutDev = 10,
    copyDev = 11,
    pasteDev = 12,
    clearDev = 13,
    cursorDev = 14
};
```

Termination Options

Specifies masks to control the timing of application termination during system shutdown or restart.

```
enum {
    kQuitBeforeNormalTimeMask = 1,
    kQuitAtNormalTimeMask = 2,
    kQuitBeforeFBAsQuitMask = 4,
    kQuitBeforeShellQuitsMask = 8,
    kQuitBeforeTerminatorAppQuitsMask = 16,
    kQuitNeverMask = 32,
    kQuitOptionsMask = 0x7F,
    kQuitNotQuitDuringInstallMask = 0x0100,
    kQuitNotQuitDuringLogoutMask = 0x0200
};
```

Discussion
Applications and background applications can control when they are asked to quit by the system at restart and shutdown by setting these bits in a `'quit'(0)` resource located in the resource fork.

Applications without this resource are terminated at `kQuitAtNormalTime`.

Availability
Available in CarbonLib 1.0 and later. Not available in Mac OS X version 10.0 and later.

Front Process Options

Specifies options for bringing windows forward when a process is activated.

```
enum {
    kSetFrontProcessFrontWindowOnly = (1 << 0)
};
```

Constants

`kSetFrontProcessFrontWindowOnly`
Activate the process, but bring only the frontmost non-floating window forward.

Available in Mac OS X version 10.2 and later.

Declared in `Processes.h`.

Launch Options

Specifies the valid launch options in the `launchControlFlags` field of the launch parameter block.

```
typedef unsigned short LaunchFlags;
enum {
  launchContinue                = 0x4000,
  launchNoFileFlags             = 0x0800,
  launchUseMinimum              = 0x0400,
  launchDontSwitch              = 0x0200,
  launchAllow24Bit              = 0x0100,
  launchInhibitDaemon           = 0x0080
};
```

Constants

`launchContinue`

Set this flag if you want your application to continue after the specified application is launched. If you do not set this flag, `LaunchApplication` terminates your application after launching the specified application, even if the launch fails.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchNoFileFlags`

Set this flag if you want the `LaunchApplication` function to ignore any value specified in the `launchFileFlags` field. If you set the `launchNoFileFlags` flag, the `LaunchApplication` function extracts the Finder flags from the application file for you. If you want to supply the file flags, clear the `launchNoFileFlags` flag and specify the Finder flags in the `launchFileFlags` field of the launch parameter block.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchUseMinimum`

Clear this flag if you want the `LaunchApplication` function to attempt to launch the application in the preferred size (as specified in the application's `'SIZE'` resource). If you set the `launchUseMinimum` flag, the `LaunchApplication` function attempts to launch the application using the largest available size greater than or equal to the minimum size but less than the preferred size. If the `LaunchApplication` function returns the result code `memFullErr` or `memFragErr`, the application cannot be launched under the current memory conditions.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchDontSwitch`

Set this flag if you do not want the launched application brought to the front. If you set this flag, the launched application runs in the background until the user brings the application to the front—for example, by clicking in one of the application's windows. Note that most applications expect to be launched in the foreground. If you clear the `launchDontSwitch` flag, the launched application is brought to the front, and your application is sent to the background.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchAllow24Bit`

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`launchInhibitDaemon`
Set this flag if you do not want `LaunchApplication` to launch a background-only application. (A background-only application has the `onlyBackground` flag set in its `'SIZE'` resource.)

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

Discussion
For more information, see `LaunchApplication` (page 1451) and `LaunchParamBlockRec` (page 1457).

Process Mode Flags

Specifies the type of information returned in a process information record.

```
enum {
    modeReserved = 0x01000000,
    modeControlPanel = 0x00080000,
    modeLaunchDontSwitch = 0x00040000,
    modeDeskAccessory = 0x00020000,
    modeMultiLaunch = 0x00010000,
    modeNeedSuspendResume = 0x00004000,
    modeCanBackground = 0x00001000,
    modeDoesActivateOnFGSwitch = 0x00000800,
    modeOnlyBackground = 0x00000400,
    modeGetFrontClicks = 0x00000200,
    modeGetAppDiedMsg = 0x00000100,
    mode32BitCompatible = 0x00000080,
    modeHighLevelEventAware = 0x00000040,
    modeLocalAndRemoteHLEvents = 0x00000020,
    modeStationeryAware = 0x00000010,
    modeUseTextEditServices = 0x00000008,
    modeDisplayManagerAware = 0x00000004
};
```

Discussion
These constants indicate, in the `processMode` field of the `ProcessInfoRec` (page 1459) structure, whether the process is an application or a desk accessory. If the process is an application, these flags return information about the application's `'SIZE'` resource.

Process Identification Constants

Specifies constants used instead of a process serial number to identify a process.

```
enum {
    kNoProcess = 0,
    kSystemProcess = 1,
    kCurrentProcess = 2
};
```

Constants
`kNoProcess`
Identifies a process that doesn't exist.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`kSystemProcess`
Identifies a process that belongs to the Operating System.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

`kCurrentProcess`
Identifies the current process.

Available in Mac OS X v10.0 and later.

Declared in `Processes.h`.

Discussion
If you want to use these constants to specify a process, you must populate a process serial number structure (`ProcessSerialNumber` (page 1461)), passing 0 in the `highLongOfPSN` field and the appropriate constant (such as `kCurrentProcess`) in the `lowLongOfPSN`. For example, to bring the current process forward, you can use the following code:

```
ProcessSerialNumber psn = { 0, kCurrentProcess };
SetFrontProcess( &psn );
```

Process Transformation Constant

Specify tranformation types to be applied when calling `TransformProcessType` (page 1456).

```
enum {
  kProcessTransformToForegroundApplication = 1L
};
typedef UInt32 ProcessApplicationTransformState;
```

Constants
`kProcessTransformToForegroundApplication`
Use to convert a background-only application to a foreground application.

Available in Mac OS X v10.3 and later.

Declared in `Processes.h`.

Result Codes

The table below lists the most common result codes returned by the Process Manager.

Result Code	Value	Description
`procNotFound`	-600	No eligible process with specified process serial number. Available in Mac OS X v10.0 and later.
`memFragErr`	-601	Not enough room to launch application with special requirements. Available in Mac OS X v10.0 and later.
`appModeErr`	-602	Addressing mode is 32-bit, but application is not 32-bit clean. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
protocolErr	-603	app made module calls in improper order Available in Mac OS X v10.0 and later.
hardwareConfigErr	-604	Hardware configuration not supported. Available in Mac OS X v10.0 and later.
appMemFullErr	-605	Partition size specified in SIZE resource is not big enough for launch. Available in Mac OS X v10.0 and later.
appIsDaemon	-606	Application runs in background only. Available in Mac OS X v10.0 and later.
wrongApplicationPlatform	-875	The application could not launch because the required platform is not available. Available in Mac OS X v10.0 and later.
appVersionTooOld	-876	The application's creator and version are incompatible with the current version of Mac OS. Available in Mac OS X v10.0 and later.
notAppropriateForClassic	-877	This application will not (or should not) run in Classic. Available in Mac OS X v10.0 and later.

Quartz Display Services Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGDirectDisplay.h CGDirectPalette.h CGDisplayConfiguration.h CGDisplayFade.h CGError.h CGRemoteOperation.h CGSession.h CGWindowLevel.h
Companion guide	Quartz Display Services Programming Topics

Overview

Note: This document was previously titled *Quartz Services Reference*. Some information related to low-level events has been moved from this document into *Quartz Event Services Reference*.

Quartz Display Services provides direct access to certain low-level features in the Mac OS X window server related to the configuration and control of display hardware. For example, you can use Quartz Display Services to:

- Examine and change display mode properties such as width, height, and pixel depth
- Configure a set of displays in a single operation
- Capture one or more displays for exclusive use
- Perform fade effects
- Activate display mirroring
- Configure gamma color correction tables and color palettes
- Receive notification of screen update operations

Functions by Task

Finding Displays

`CGMainDisplayID` (page 1522)
Returns the display ID of the main display.

`CGGetOnlineDisplayList` (page 1521)
Provides a list of displays that are online (active, mirrored, or sleeping).

`CGGetActiveDisplayList` (page 1515)
Provides a list of displays that are active (or drawable).

`CGGetDisplaysWithOpenGLDisplayMask` (page 1516)
Provides a list of displays that corresponds to the bits set in an OpenGL display mask.

`CGGetDisplaysWithPoint` (page 1517)
Provides a list of online displays with bounds that include the specified point.

`CGGetDisplaysWithRect` (page 1518)
Gets a list of online displays with bounds that intersect the specified rectangle.

`CGOpenGLDisplayMaskToDisplayID` (page 1522)
Maps an OpenGL display mask to a display ID.

`CGDisplayIDToOpenGLDisplayMask` (page 1498)
Maps a display ID to an OpenGL display mask.

Capturing and Releasing Displays

`CGDisplayCapture` (page 1493)
Captures a display for exclusive use by an application.

`CGDisplayCaptureWithOptions` (page 1493)
Captures a display for exclusive use by an application, using the specified options.

`CGDisplayRelease` (page 1507)
Releases a captured display.

`CGDisplayIsCaptured` (page 1501)
Returns a Boolean value indicating whether a display is captured.

`CGCaptureAllDisplays` (page 1478)
Captures all attached displays.

`CGCaptureAllDisplaysWithOptions` (page 1479)
Captures all attached displays, using the specified options.

`CGReleaseAllDisplays` (page 1529)
Releases all captured displays.

`CGShieldingWindowID` (page 1535)
Returns the window ID of the shield window for a captured display.

`CGShieldingWindowLevel` (page 1536)
Returns the window level of the shield window for a captured display.

`CGDisplayAddressForPosition` (page 1485)
Returns the address in frame buffer memory that corresponds to a position on an online display.

`CGDisplayBaseAddress` (page 1487)
Returns the base address in frame buffer memory of an online display.

`CGDisplayGetDrawingContext` (page 1497)
Returns a graphics context suitable for drawing to a captured display.

Configuring Displays

`CGBeginDisplayConfiguration` (page 1477)
Begins a new set of display configuration changes.

`CGCancelDisplayConfiguration` (page 1478)
Cancels a set of display configuration changes.

`CGCompleteDisplayConfiguration` (page 1479)
Completes a set of display configuration changes.

`CGConfigureDisplayMirrorOfDisplay` (page 1481)
Changes the configuration of a mirroring set.

`CGConfigureDisplayMode` (page 1482)
Configures the display mode of a display.

`CGConfigureDisplayOrigin` (page 1483)
Configures the origin of a display in global display (desktop) coordinates.

`CGRestorePermanentDisplayConfiguration` (page 1531)
Restores the permanent display configuration settings for the current user.

`CGConfigureDisplayStereoOperation` (page 1484)
Enables or disables stereo operation for a display, as part of a display configuration.

`CGDisplaySetStereoOperation` (page 1511)
Immediately enables or disables stereo operation for a display.

Getting the Display Configuration

`CGDisplayCopyColorSpace` (page 1494)
Returns the color space for a display.

`CGDisplayIOServicePort` (page 1498)
Returns the I/O Kit service port of the specified display.

`CGDisplayIsActive` (page 1499)
Returns a Boolean value indicating whether a display is active.

`CGDisplayIsAlwaysInMirrorSet` (page 1499)
Returns a Boolean value indicating whether a display is always in a mirroring set.

`CGDisplayIsAsleep` (page 1500)
Returns a Boolean value indicating whether a display is sleeping (and is therefore not drawable.)

`CGDisplayIsBuiltin` (page 1500)
Returns a Boolean value indicating whether a display is built-in, such as the internal display in portable systems.

`CGDisplayIsInHWMirrorSet` (page 1501)
Returns a Boolean value indicating whether a display is in a hardware mirroring set.

`CGDisplayIsInMirrorSet` (page 1502)
Returns a Boolean value indicating whether a display is in a mirroring set.

`CGDisplayIsMain` (page 1502)
Returns a Boolean value indicating whether a display is the main display.

`CGDisplayIsOnline` (page 1503)
Returns a Boolean value indicating whether a display is connected or online.

`CGDisplayIsStereo` (page 1503)
Returns a Boolean value indicating whether a display is running in a stereo graphics mode.

`CGDisplayMirrorsDisplay` (page 1504)
For a secondary display in a mirroring set, returns the primary display.

`CGDisplayModelNumber` (page 1504)
Returns the model number of a display monitor.

`CGDisplayPrimaryDisplay` (page 1506)
Returns the primary display in a hardware mirroring set.

`CGDisplayRotation` (page 1508)
Returns the rotation angle of a display in degrees.

`CGDisplayScreenSize` (page 1509)
Returns the width and height of a display in millimeters.

`CGDisplaySerialNumber` (page 1510)
Returns the serial number of a display monitor.

`CGDisplayUnitNumber` (page 1513)
Returns the logical unit number of a display.

`CGDisplayUsesOpenGLAcceleration` (page 1514)
Returns a Boolean value indicating whether Quartz is using OpenGL-based window acceleration (Quartz Extreme) to render in a display.

`CGDisplayVendorNumber` (page 1514)
Returns the vendor number of the specified display's monitor.

Registering for Notification of Display Configuration Changes

These functions are used to register and unregister a callback function for notification of display configuration changes.

`CGDisplayRegisterReconfigurationCallback` (page 1507)
Registers a callback function to be invoked whenever a local display is reconfigured.

`CGDisplayRemoveReconfigurationCallback` (page 1508)
Removes the registration of a callback function that's invoked whenever a local display is reconfigured.

Retrieving Display Parameters

`CGDisplayBounds` (page 1491)
Returns the bounds of a display in global display space.

`CGDisplayPixelsHigh` (page 1505)
Returns the display height in pixel units.

`CGDisplayPixelsWide` (page 1506)
Returns the display width in pixel units.

`CGDisplayBitsPerPixel` (page 1491)
Returns the number of bits used to represent a pixel in the frame buffer.

`CGDisplayBitsPerSample` (page 1491)
Returns the number of bits used to represent a pixel component in the frame buffer.

`CGDisplaySamplesPerPixel` (page 1509)
Returns the number of color components used to represent a pixel.

`CGDisplayBytesPerRow` (page 1492)
Returns the number of bytes per row in a display.

Using Display Modes

`CGDisplayAvailableModes` (page 1486)
Returns information about the currently available display modes.

`CGDisplayBestModeForParameters` (page 1488)
Returns information about the display mode closest to a specified depth and screen size.

`CGDisplayBestModeForParametersAndRefreshRate` (page 1488)
Returns information about the display mode closest to a specified depth, screen size, and refresh rate.

`CGDisplayBestModeForParametersAndRefreshRateWithProperty` (page 1490)
Returns information about the display mode closest to a specified depth, screen size, and refresh rate, with a required property.

`CGDisplayCurrentMode` (page 1494)
Returns information about the current display mode.

`CGDisplaySwitchToMode` (page 1512)
Switches a display to a different mode.

Adjusting the Display Gamma

`CGSetDisplayTransferByFormula` (page 1533)
Sets the gamma function for a display, by specifying the coefficients of the gamma transfer formula.

`CGGetDisplayTransferByFormula` (page 1518)
Gets the coefficients of the gamma transfer formula for a display.

`CGSetDisplayTransferByTable` (page 1535)
Sets the color gamma function for a display, by specifying the values in the RGB gamma tables.

`CGGetDisplayTransferByTable` (page 1520)
Gets the values in the RGB gamma tables for a display.

`CGSetDisplayTransferByByteTable` (page 1532)
Sets the byte values in the 8-bit RGB gamma tables for a display.

`CGDisplayRestoreColorSyncSettings` (page 1508)
Restores the gamma tables to the values in the user's ColorSync display profile.

`CGDisplayGammaTableCapacity` (page 1497)
Returns the capacity, or number of entries, in the gamma table for a display.

Working With Color Palettes

`CGPaletteCreateDefaultColorPalette` (page 1523)
Returns a new display palette representing the default 8-bit color palette.

`CGPaletteCreateFromPaletteBlendedWithColor` (page 1524)
Returns a new tinted display palette. The new palette is derived from an existing palette blended with a solid color, at a specified level of intensity.

`CGPaletteCreateWithByteSamples` (page 1524)
Returns a new display palette using 8-bit sample data.

`CGPaletteCreateWithCapacity` (page 1525)
Returns a new display palette with a specified capacity. The new palette is initialized from the default color palette.

`CGPaletteCreateWithDisplay` (page 1525)
Returns a copy of the current palette for a display.

`CGPaletteCreateWithSamples` (page 1525)
Returns a new display palette using RGB sample data.

`CGPaletteCreateCopy` (page 1523)
Returns a copy of a specified display palette.

`CGPaletteRelease` (page 1528)
Decrements the retain count of a display palette.

`CGPaletteGetColorAtIndex` (page 1526)
Returns the color value at the specified index.

`CGPaletteGetIndexForColor` (page 1526)
Returns the index of the display palette entry that most closely matches a specified color value.

`CGPaletteGetNumberOfSamples` (page 1527)
Returns the number of colors in a display palette.

`CGPaletteIsEqualToPalette` (page 1527)
Returns a Boolean value indicating whether two display palettes are equal.

`CGPaletteSetColorAtIndex` (page 1528)
Updates the color value at the specified index in a display palette.

`CGDisplayCanSetPalette` (page 1492)
Returns a Boolean value indicating whether the current display mode supports palettes.

`CGDisplaySetPalette` (page 1510)
Sets the palette for a display.

Display Fade Effects

`CGConfigureDisplayFadeEffect` (page 1480)
Modifies the settings of the built-in fade effect that occurs during a display configuration.

`CGAcquireDisplayFadeReservation` (page 1476)
Reserves the fade hardware for a specified time interval.

`CGDisplayFade` (page 1495)
Performs a single fade operation.

`CGDisplayFadeOperationInProgress` (page 1496)
Returns a Boolean value indicating whether a fade operation is currently in progress.

`CGReleaseDisplayFadeReservation` (page 1530)
Releases a display fade reservation, and unfades the display if needed.

Beam Position

These functions are advisory in nature and depend on IO Kit and hardware-specific drivers to implement support. If you need extremely precise timing, or access to vertical blanking interrupts, you should consider writing a device driver to tie into hardware-specific capabilities.

`CGDisplayBeamPosition` (page 1487)
Returns the current beam position on a display.

`CGDisplayWaitForBeamPositionOutsideLines` (page 1515)
Waits until the beam position moves outside a region in a display screen. This function is not designed for VBL drawing synchronization.

Controlling the Mouse Cursor

`CGDisplayHideCursor` (page 1497)
Hides the mouse cursor, and increments the hide cursor count.

`CGDisplayShowCursor` (page 1512)
Decrements the hide cursor count, and shows the mouse cursor if the count is zero.

`CGDisplayMoveCursorToPoint` (page 1505)
Moves the mouse cursor to a specified point relative to the display origin (the upper left corner of the display).

`CGCursorIsVisible` (page 1485)
Returns a Boolean value indicating whether the mouse cursor is visible.

`CGCursorIsDrawnInFramebuffer` (page 1484)
Returns a Boolean value indicating whether the mouse cursor is drawn in frame buffer memory.

`CGAssociateMouseAndMouseCursorPosition` (page 1477)
Connects or disconnects the mouse and cursor while an application is in the foreground.

`CGWarpMouseCursorPosition` (page 1539)
Moves the mouse cursor without generating events.

`CGGetLastMouseDelta` (page 1520)
Reports the change in mouse position since the last mouse movement event received by the application.

Getting Window Server Information

`CGSessionCopyCurrentDictionary` (page 1532)
Returns information about the caller's window server session.

`CGWindowServerCFMachPort` (page 1540)
Returns a Core Foundation mach port (CFMachPort) that corresponds to the Mac OS X window server.

`CGWindowLevelForKey` (page 1539)
Returns the window level that corresponds to one of the standard window types.

Getting Information About Refresh and Move Operations

You can use these functions to find out what areas on local displays are changing their appearance as the result of operations such as drawing, window movement or scrolling, and display reconfiguration.

`CGRegisterScreenRefreshCallback` (page 1529)
Registers a callback function to be invoked when local displays are refreshed or modified.

`CGUnregisterScreenRefreshCallback` (page 1536)
Removes a previously registered callback function invoked when local displays are refreshed or modified.

`CGWaitForScreenRefreshRects` (page 1537)
Waits for screen refresh operations.

`CGScreenRegisterMoveCallback` (page 1531)
Registers a callback function to be invoked when an area of the display is moved.

`CGScreenUnregisterMoveCallback` (page 1532)
Removes a previously registered callback function invoked when an area of the display is moved.

`CGWaitForScreenUpdateRects` (page 1538)
Waits for screen update operations.

`CGReleaseScreenRefreshRects` (page 1530)
Deallocates a list of rectangles that represent changed areas on local displays.

Functions

CGAcquireDisplayFadeReservation

Reserves the fade hardware for a specified time interval.

```
CGError CGAcquireDisplayFadeReservation (
   CGDisplayReservationInterval seconds,
   CGDisplayFadeReservationToken *pNewToken
);
```

Parameters

seconds
The desired number of seconds to reserve the fade hardware. An application can specify any value in the interval `(0, kCGMaxDisplayReservationInterval]`.

pNewToken
A pointer to storage (provided by the caller) for a fade reservation token. On return, the storage contains a new token.

Return Value
Returns `kCGErrorNoneAvailable` if another fade reservation is in effect. Otherwise, returns `kCGErrorSuccess`.

Discussion
Before performing a fade operation, an application must reserve the fade hardware for a specified period of time. Quartz returns a token that represents a new fade reservation. The application uses this token as an argument in subsequent calls to other display fade functions.

During the fade reservation interval, the application has exclusive rights to use the fade hardware. At the end of the interval, the token becomes invalid and the hardware automatically returns to a normal state. Typically the application calls `CGReleaseDisplayFadeReservation` (page 1530) to release the fade reservation before it expires.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGAssociateMouseAndMouseCursorPosition

Connects or disconnects the mouse and cursor while an application is in the foreground.

```
CGError CGAssociateMouseAndMouseCursorPosition (
   boolean_t connected
);
```

Parameters

connected
Pass `true` if the mouse and cursor should be connected; otherwise, pass `false`.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
When you call this function to disconnect the cursor and mouse, all events received by your application have a constant absolute location but contain mouse delta (change in X and Y) data. You may hide the cursor or change it into something appropriate for your application. You can reposition the cursor by using the function `CGDisplayMoveCursorToPoint` (page 1505) or the function `CGWarpMouseCursorPosition` (page 1539).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGRemoteOperation.h`

CGBeginDisplayConfiguration

Begins a new set of display configuration changes.

```
CGError CGBeginDisplayConfiguration (
   CGDisplayConfigRef *pConfigRef
);
```

Parameters

pConfigRef

A pointer to storage you provide for a display configuration. On return, your storage contains a new display configuration.

Return Value
A result code. If the object is successfully created, the result is `kCGErrorSuccess`. For other possible values, see "Quartz Display Services Result Codes" (page 1564).

Discussion
This function creates a display configuration object that provides a context for a set of display configuration changes. After you specify the desired changes, you use `CGCompleteDisplayConfiguration` (page 1479) to apply them in a single transaction.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGCancelDisplayConfiguration

Cancels a set of display configuration changes.

```
CGError CGCancelDisplayConfiguration (
   CGDisplayConfigRef configRef
);
```

Parameters

configRef

The display configuration to cancel. On return, the configuration is cancelled and is no longer valid.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
This function is used to abandon a display configuration. As a side effect, the display configuration object is released.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGCaptureAllDisplays

Captures all attached displays.

```
CGDisplayErr CGCaptureAllDisplays (
   void
);
```

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
This function captures all attached displays in a single operation. This operation provides an immersive environment for your application, and it prevents other applications from trying to adjust to display changes.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGDisplayCapture` (page 1493)

Declared In
`CGDirectDisplay.h`

CGCaptureAllDisplaysWithOptions

Captures all attached displays, using the specified options.

```
CGDisplayErr CGCaptureAllDisplaysWithOptions (
   CGCaptureOptions options
);
```

Parameters

options
The options to use. See "Display Capture Options" (page 1553).

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
This function allows you to specify one or more options to use during capture of all attached displays.

Availability
Available in Mac OS X v10.3 and later.

See Also
`CGCaptureAllDisplays` (page 1478)

Declared In
`CGDirectDisplay.h`

CGCompleteDisplayConfiguration

Completes a set of display configuration changes.

```
CGError CGCompleteDisplayConfiguration (
   CGDisplayConfigRef configRef,
   CGConfigureOption option
);
```

Parameters

configRef

The display configuration with the desired changes. On return, this configuration is no longer valid.

option

The scope of the display configuration changes. Pass one of the constants listed in "Display Configuration Scopes" (page 1555).

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

This function applies a set of display configuration changes as a single atomic transaction. The duration or scope of the changes depends on the value of the `option` parameter. The possible scopes are fully described in "Display Configuration Scopes" (page 1555).

A configuration change may fail if an unsupported display mode is requested, or if another application is running in full-screen mode.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGDisplayConfiguration.h`

CGConfigureDisplayFadeEffect

Modifies the settings of the built-in fade effect that occurs during a display configuration.

```
CGError CGConfigureDisplayFadeEffect (
   CGDisplayConfigRef configRef,
   CGDisplayFadeInterval fadeOutSeconds,
   CGDisplayFadeInterval fadeInSeconds,
   float fadeRed,
   float fadeGreen,
   float fadeBlue
);
```

Parameters

configRef

A display configuration, acquired by calling `CGBeginDisplayConfiguration` (page 1477).

fadeOutSeconds

The time in seconds to fade from the normal display to the specified fade color. The fade out is completed before the display configuration is changed. If the interval is 0, Quartz applies the color immediately.

fadeInSeconds

Time in seconds to return from the specified fade color to the normal display. The fade-in is run asynchronously after the display configuration is changed.

fadeRed
An intensity value in the interval [0, 1] that represents the red component of the desired blend color.

fadeGreen
An intensity value in the interval [0, 1] that represents the green component of the desired blend color.

fadeBlue
An intensity value in the interval [0, 1] that represents the blue component of the desired blend color.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
This function provides a way to customize the built-in fade effect that Quartz performs when displays are reconfigured. The default time settings for this fade effect are 0.3 seconds to fade out, and 0.5 seconds to fade back in. The default fade color is French Blue for a normal desktop, and black for a captured display.

Before using this function, you need to call `CGBeginDisplayConfiguration` (page 1477) to acquire the display configuration token for the desired display. No fade reservation is needed—when you call `CGCompleteDisplayConfiguration` (page 1479), Quartz reserves the fade hardware (assuming it is available) and performs the fade.

Calling this function modifies the fade behavior for a single display configuration, and has no permanent effect.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGConfigureDisplayMirrorOfDisplay

Changes the configuration of a mirroring set.

```
CGError CGConfigureDisplayMirrorOfDisplay (
   CGDisplayConfigRef configRef,
   CGDirectDisplayID display,
   CGDirectDisplayID masterDisplay
);
```

Parameters

configRef
A display configuration, acquired by calling `CGBeginDisplayConfiguration` (page 1477).

display
The display to add to a mirroring set.

masterDisplay
A display in a mirroring set, or `kCGNullDirectDisplay` to disable mirroring. To specify the main display, use `CGMainDisplayID` (page 1522).

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
Display mirroring and display matte generation are implemented either in hardware (preferred) or software, at the discretion of the device driver.

- Hardware mirroring

  With hardware mirroring enabled, all drawing is directed to the primary display—see `CGDisplayPrimaryDisplay` (page 1506).

  If the device driver selects hardware matte generation, the display bounds and rowbytes values are adjusted to reflect the active drawable area.

- Software mirroring

  In this form of mirroring, identical content is drawn into each display in the mirroring set. Applications that use the window system need not be concerned about mirroring, as the window system takes care of all flushing of window content to the appropriate displays.

  Applications that draw directly to the display, as with display capture, must make sure to draw the same content to all mirrored displays in a software mirror set. When drawing to software mirrored displays using a full screen OpenGL context (not drawing through a window), you should create shared OpenGL contexts for each display and re-render for each display.

You can use the function `CGGetActiveDisplayList` (page 1515) to determine which displays are active, or drawable. This automatically gives your application the correct view of the current displays.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGConfigureDisplayMode

Configures the display mode of a display.

```
CGError CGConfigureDisplayMode (
   CGDisplayConfigRef configRef,
   CGDirectDisplayID display,
   CFDictionaryRef mode
);
```

Parameters

configRef
A display configuration, acquired by calling `CGBeginDisplayConfiguration` (page 1477).

display
The display being configured.

mode
A display mode dictionary (see the discussion below).

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
A display mode is a set of properties such as width, height, pixel depth, and refresh rate, and options such as stretched LCD panel filling.

The display mode you provide must be one of the following:

- A dictionary returned by one of the `CGDisplayBestMode` functions, such as `CGDisplayBestModeForParameters` (page 1488).
- A dictionary in the array returned by `CGDisplayAvailableModes` (page 1486).

If you use this function to change the mode of a display in a mirroring set, Quartz may adjust the bounds, resolutions, and depth of the other displays in the set to a safe mode, with matching depth and the smallest enclosing size.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGConfigureDisplayOrigin

Configures the origin of a display in global display (desktop) coordinates.

```
CGError CGConfigureDisplayOrigin (
   CGDisplayConfigRef configRef,
   CGDirectDisplayID display,
   CGDisplayCoord x,
   CGDisplayCoord y
);
```

Parameters

configRef
A display configuration, acquired by calling `CGBeginDisplayConfiguration` (page 1477).

display
The display being configured.

x
The desired x-coordinate for the upper left corner of the display.

y
The desired y-coordinate for the upper left corner of the display.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
In Quartz, the upper left corner of a display is called the origin. The origin of a display is always specified in global display (desktop) coordinates. The origin of the main or primary display is (0,0).

The new origin is placed as close as possible to the requested location, without overlapping or leaving a gap between displays.

If you use this function to change the origin of a mirrored display, the display may be removed from the mirroring set.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGDisplayConfiguration.h

CGConfigureDisplayStereoOperation

Enables or disables stereo operation for a display, as part of a display configuration.

```
CGError CGConfigureDisplayStereoOperation (
   CGDisplayConfigRef configRef,
   CGDirectDisplayID display,
   boolean_t stereo,
   boolean_t forceBlueLine
);
```

Parameters

configRef
A display configuration, acquired by calling `CGBeginDisplayConfiguration` (page 1477).

display
The display being configured.

stereo
Pass `true` if you want to enable stereo operation. To disable it, pass `false`.

forceBlueLine
When in stereo operation, a display may need to generate a special stereo sync signal as part of the video output. The sync signal consists of a blue line which occupies the first 25% of the last scanline for the left eye view, and the first 75% of the last scanline for the right eye view. The remainder of the scanline is black. To force the display to generate this sync signal, pass `true`; otherwise, pass `false`.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
The system normally detects the presence of a stereo window and automatically switches a display containing a stereo window to stereo operation. This function provides a mechanism to force a display to stereo operation, and to set options (blue line sync signal) when in stereo operation.

On success, the display resolution, mirroring mode, and available display modes may change due to hardware-specific capabilities and limitations. You should check these settings to verify that they are appropriate for your application.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGDisplayConfiguration.h

CGCursorIsDrawnInFramebuffer

Returns a Boolean value indicating whether the mouse cursor is drawn in frame buffer memory.

```
boolean_t CGCursorIsDrawnInFramebuffer (
   void
);
```

Return Value
If `true`, the cursor is drawn in frame buffer memory; otherwise, `false`.

Discussion
This function returns a Boolean value that indicates whether or not the cursor is drawn in the frame buffer. (The cursor could exist in an overlay plane or a similar mechanism that puts pixels on-screen without altering frame buffer content.) If the cursor is drawn in the frame buffer, it is read back along with window data.

The reported Boolean value is based on the union of the state of the cursor on all displays. If the cursor is drawn in the frame buffer on any display, the function returns `true`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGRemoteOperation.h`

CGCursorIsVisible

Returns a Boolean value indicating whether the mouse cursor is visible.

```
boolean_t CGCursorIsVisible (
   void
);
```

Return Value
If `true`, the cursor is visible on any display; otherwise, `false`.

Discussion
To hide or show the cursor, you can use the functions `CGDisplayHideCursor` (page 1497) and `CGDisplayShowCursor` (page 1512).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGRemoteOperation.h`

CGDisplayAddressForPosition

Returns the address in frame buffer memory that corresponds to a position on an online display.

```
void * CGDisplayAddressForPosition (
   CGDirectDisplayID display,
   CGDisplayCoord x,
   CGDisplayCoord y
);
```

Parameters

display

The display to access.

x

The x-coordinate of a position in global display space. The origin is the upper left corner of the main display.

y

The y-coordinate of a position in global display space. The origin is the upper left corner of the main display, and the y-axis is oriented down.

Return Value

The address in frame buffer memory that corresponds to the specified position. If the display ID is invalid or the point lies outside the bounds of the display, the return value is `NULL`.

Discussion

If the display has not been captured, the returned address may refer to read-only memory.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayAvailableModes

Returns information about the currently available display modes.

```
CFArrayRef CGDisplayAvailableModes (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

An array of dictionaries with display mode information, or `NULL` if the display is invalid. The array is owned by the system and you should not release it. Each dictionary in the array contains information about a mode that the display supports. For a list of the properties in a display mode dictionary, see "Display Mode Standard Properties" (page 1557) and "Display Mode Optional Properties" (page 1558). For general information about using dictionaries, see *CFDictionary Reference*.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

Declared In
CGDirectDisplay.h

CGDisplayBaseAddress

Returns the base address in frame buffer memory of an online display.

```
void * CGDisplayBaseAddress (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The base address in frame buffer memory of the specified display. If the display ID is invalid, the return value is NULL.

Discussion
If the display has not been captured, the returned address may refer to read-only memory.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGDisplayBeamPosition

Returns the current beam position on a display.

```
CGBeamPosition CGDisplayBeamPosition (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The current beam position on the specified display. If the display does not implement conventional video vertical and horizontal sweep in painting, or the driver does not implement this functionality, 0 is returned.

Discussion
This function returns the number of the scan line on which the beam is currently positioned, expressed as a non-negative integer. The value increases as the beam moves lower on the display.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGDisplayBestModeForParameters

Returns information about the display mode closest to a specified depth and screen size.

```
CFDictionaryRef CGDisplayBestModeForParameters (
   CGDirectDisplayID display,
   size_t bitsPerPixel,
   size_t width,
   size_t height,
   boolean_t *exactMatch
);
```

Parameters

display

The display to optimize.

bitsPerPixel

Optimal display depth in bits per pixel. Note that this value is not the same as pixel depth, which is the number of bits per channel or component.

width

Optimal display width in pixel units.

height

Optimal display height in pixel units.

exactMatch

A pointer to a Boolean variable. On return, its value is `true` if an exact match in display depth, width, and height is found; otherwise, `false`. If this information is not needed, pass `NULL`.

Return Value

A display mode dictionary, or `NULL` if the display is invalid. The dictionary is owned by the system and you should not release it. The dictionary contains information about the display mode closest to the specified depth and screen size. For a list of the properties in a display mode dictionary, see "Display Mode Standard Properties" (page 1557) and "Display Mode Optional Properties" (page 1558). For general information about using dictionaries, see *CFDictionary Reference*.

Discussion

This function tries to find an optimal display mode for the specified display. The function first tries to find a mode with the specified pixel depth and dimensions equal to or greater than the specified width and height. If no depth match is found, it tries to find a mode with greater depth and the same or greater dimensions. If a suitable display mode is not found, this function simply returns the current display mode.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayBestModeForParametersAndRefreshRate

Returns information about the display mode closest to a specified depth, screen size, and refresh rate.

```
CFDictionaryRef CGDisplayBestModeForParametersAndRefreshRate (
   CGDirectDisplayID display,
   size_t bitsPerPixel,
   size_t width,
   size_t height,
   CGRefreshRate refresh,
   boolean_t *exactMatch
);
```

Parameters

display

The display to access.

bitsPerPixel

Optimal display depth, in bits per pixel. Note that this value is not the same as pixel depth, which is the number of bits per channel or component.

width

Optimal display width, in pixel units.

height

Optimal display height, in pixel units.

refresh

Optimal display refresh rate, in frames per second.

exactMatch

A pointer to a Boolean variable. On return, its value is `true` if an exact match in display depth, width, height, and refresh rate is found; otherwise, `false`. If this information is not needed, pass `NULL`.

Return Value

A display mode dictionary, or `NULL` if the display is invalid. The dictionary is owned by the system and you should not release it. The dictionary contains information about the display mode closest to the specified depth, screen size, and refresh rate. For a list of the properties in a display mode dictionary, see "Display Mode Standard Properties" (page 1557) and "Display Mode Optional Properties" (page 1558). For general information about using dictionaries, see *CFDictionary Reference*.

Discussion

This function searches the list of available display modes for a mode that comes closest to satisfying these criteria:

- Has a pixel depth equal to or greater than the specified depth
- Has dimensions equal to or greater than the specified height and width
- Uses a refresh rate equal to or near the specified rate

If a suitable display mode is not found, this function simply returns the current display mode.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayBestModeForParametersAndRefreshRateWithProperty

Returns information about the display mode closest to a specified depth, screen size, and refresh rate, with a required property.

```
CFDictionaryRef CGDisplayBestModeForParametersAndRefreshRateWithProperty (
   CGDirectDisplayID display,
   size_t bitsPerPixel,
   size_t width,
   size_t height,
   CGRefreshRate refresh,
   CFStringRef property,
   boolean_t *exactMatch
);
```

Parameters

display

The display to access.

bitsPerPixel

Optimal display depth, in bits per pixel. Note that this value is not the same as pixel depth, which is the number of bits per channel or component.

width

Optimal display width, in pixels.

height

Optimal display height, in pixels.

refresh

Optimal display refresh rate, in refreshes per second.

property

A required display mode property. For a list of the properties you can specify, see "Display Mode Optional Properties" (page 1558).

exactMatch

A pointer to a Boolean variable. On return, its value is `true` if an exact match in display depth, width, height, refresh rate, and property is found; otherwise, `false`. If this information is not needed, pass `NULL`.

Return Value

A display mode dictionary, or `NULL` if the display is invalid. The dictionary is owned by the system and you should not release it. The dictionary contains information about the display mode with the specified property that comes closest to the specified depth, screen size, and refresh rate. For a list of the properties in a display mode dictionary, see "Display Mode Standard Properties" (page 1557) and "Display Mode Optional Properties" (page 1558). For general information about using dictionaries, see *CFDictionary Reference*.

Discussion

This function searches the list of available display modes for a mode that includes the specified property and comes closest to satisfying these criteria:

- Has a pixel depth equal to or greater than the specified depth
- Has dimensions equal to or greater than the specified height and width
- Uses a refresh rate equal to or near the specified rate

If no matching display mode is found, this function simply returns the current display mode.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayBitsPerPixel

Returns the number of bits used to represent a pixel in the frame buffer.

```
size_t CGDisplayBitsPerPixel (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The number of bits used to represent a pixel in the frame buffer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayBitsPerSample

Returns the number of bits used to represent a pixel component in the frame buffer.

```
size_t CGDisplayBitsPerSample (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The number of bits used to represent a pixel component such as a color value in the frame buffer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayBounds

Returns the bounds of a display in global display space.

```
CGRect CGDisplayBounds (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

The bounds of the display, expressed as a rectangle in the global display coordinate space (relative to the upper left corner of the main display).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayBytesPerRow

Returns the number of bytes per row in a display.

```
size_t CGDisplayBytesPerRow (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

The number of bytes per row in the display. This number also represents the stride between pixels in the same column of the display.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayCanSetPalette

Returns a Boolean value indicating whether the current display mode supports palettes.

```
boolean_t CGDisplayCanSetPalette (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

If `true`, the current display mode supports palettes; otherwise, `false`.

Discussion
Palettes are supported in any display selected to run in a 256-color display mode.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayCapture

Captures a display for exclusive use by an application.

```
CGDisplayErr CGDisplayCapture (
   CGDirectDisplayID display
);
```

Parameters

display
The display to capture.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
When an application captures a display, Quartz does not allow other applications and system services to use the display or change its configuration.

If hardware or software mirroring is in effect, the easiest way to capture the primary display and all mirrored displays is to use the function `CGCaptureAllDisplays` (page 1478). In case of software mirroring, applications that draw directly to the display must make sure to draw the same content to all displays in the mirror set.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayCaptureWithOptions

Captures a display for exclusive use by an application, using the specified options.

```
CGDisplayErr CGDisplayCaptureWithOptions (
   CGDirectDisplayID display,
   CGCaptureOptions options
);
```

Parameters

display
The display to capture.

options
The options to use. See "Display Capture Options" (page 1553).

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
This function allows you to specify one or more options to use during capture of a display.

Availability
Available in Mac OS X v10.3 and later.

See Also
`CGDisplayCapture` (page 1493)

Declared In
`CGDirectDisplay.h`

CGDisplayCopyColorSpace

Returns the color space for a display.

```
CGColorSpaceRef CGDisplayCopyColorSpace (
    CGDirectDisplayID display
);
```

Parameters

display
The display whose color space you want to obtain.

Return Value
The current color space for the specified display. The caller is responsible for releasing the color space with the `CGColorSpaceRelease` (page 55) function.

Discussion
This function returns a display-dependent ICC-based color space. You can use this function when rendering content for a specific display in order to produce color-matched output for that display.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayCurrentMode

Returns information about the current display mode.

```
CFDictionaryRef CGDisplayCurrentMode (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
A display mode dictionary, or `NULL` if the display is invalid. The dictionary is owned by the system and you should not release it. The dictionary contains information about the current display mode. For a list of the properties in a display mode dictionary, see "Display Mode Standard Properties" (page 1557) and "Display Mode Optional Properties" (page 1558). For general information about using dictionaries, see *CFDictionary Reference*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayFade

Performs a single fade operation.

```
CGError CGDisplayFade (
   CGDisplayFadeReservationToken myToken,
   CGDisplayFadeInterval seconds,
   CGDisplayBlendFraction startBlend,
   CGDisplayBlendFraction endBlend,
   float redBlend,
   float greenBlend,
   float blueBlend,
   boolean_t synchronous
);
```

Parameters

myToken
A reservation token for the fade hardware, acquired by calling `CGAcquireDisplayFadeReservation` (page 1476).

seconds
The desired number of seconds for the fade operation. You should use a value in the interval `[0, kCGMaxDisplayReservationInterval]`. If the value is 0, the ending blend color is applied immediately.

startBlend
An intensity value in the interval [0, 1] that specifies the alpha component of the desired blend color at the beginning of the fade operation. See "Display Fade Blend Fractions" (page 1556).

endBlend
An intensity value in the interval [0, 1] that specifies the alpha component of the desired blend color at the end of the fade operation. See "Display Fade Blend Fractions" (page 1556).

redBlend
An intensity value in the interval [0, 1] that specifies the red component of the desired blend color.

greenBlend
An intensity value in the interval [0, 1] that specifies the green component of the desired blend color.

blueBlend
An intensity value in the interval [0, 1] that specifies the blue component of the desired blend color.

synchronous

Pass `true` if you want the fade operation to be synchronous; otherwise, pass `false`. If a fade operation is synchronous, the function does not return until the operation is complete.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
Over the fade operation time interval, Quartz interpolates a blending coefficient between the starting and ending values given, applying a nonlinear (sine-based) bias term. Using this coefficient, the video output is blended with the specified color.

The following example shows how to perform a two-second synchronous fade-out to black:

```
CGDisplayFade (
    myToken,
    2.0,                        // 2 seconds
    kCGDisplayBlendNormal,      // starting state
    kCGDisplayBlendSolidColor,  // ending state
    0.0, 0.0, 0.0,              // black
    true                        // wait for completion
);
```

To perform a two-second asynchronous fade-in from black:

```
CGDisplayFade (
    myToken,
    2.0,                        // 2 seconds
    kCGDisplayBlendSolidColor,  // starting state
    kCGDisplayBlendNormal,      // ending state
    0.0, 0.0, 0.0,              // black
    false                       // don't wait for completion
);
```

If you specify an asynchronous fade operation, it's safe to call `CGReleaseDisplayFadeReservation` (page 1530) immediately after this function returns.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGDisplayFadeOperationInProgress

Returns a Boolean value indicating whether a fade operation is currently in progress.

```
boolean_t CGDisplayFadeOperationInProgress (
   void
);
```

Return Value
If `true`, a fade operation is currently in progress; otherwise, `false`.

Discussion
You may call this function from any task running on the system. The calling task need not have a valid fade reservation.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGDisplayGammaTableCapacity

Returns the capacity, or number of entries, in the gamma table for a display.

```
CGTableCount CGDisplayGammaTableCapacity (
   CGDirectDisplayID display
);
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayGetDrawingContext

Returns a graphics context suitable for drawing to a captured display.

```
CGContextRef CGDisplayGetDrawingContext (
   CGDirectDisplayID display
);
```

Parameters
display
The display to access.

Return Value
A Quartz graphics context suitable for drawing to a captured display, or `NULL` if the display has not been captured. The context is owned by the system and you should not release it.

Discussion
After capturing a display or changing the configuration of a captured display, you can use this function to obtain the current graphics context for the display. The graphics context remains valid while the display is captured and the display configuration is unchanged. Releasing the captured display or reconfiguring the display invalidates the context. To determine when the display configuration is changing, you can use the function `CGDisplayRegisterReconfigurationCallback` (page 1507) to register a display reconfiguration callback.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayHideCursor

Hides the mouse cursor, and increments the hide cursor count.

```
CGDisplayErr CGDisplayHideCursor (
   CGDirectDisplayID display
);
```

Parameters

display

This parameter is not used. By default, you may pass `kCGDirectMainDisplay`.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

This function hides the cursor regardless of its current location; the `display` parameter is ignored. In most cases, the caller must be the foreground application to affect the cursor.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGDisplayShowCursor` (page 1512)

Declared In

`CGDirectDisplay.h`

CGDisplayIDToOpenGLDisplayMask

Maps a display ID to an OpenGL display mask.

```
CGOpenGLDisplayMask CGDisplayIDToOpenGLDisplayMask (
   CGDirectDisplayID display
);
```

Parameters

display

The display ID to be converted.

Return Value

The OpenGL display mask that corresponds to the specified display.

Discussion

OpenGL sometimes identifies a display using a bitmask with one bit set. This function maps a display ID to the corresponding OpenGL display mask.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayIOServicePort

Returns the I/O Kit service port of the specified display.

```
io_service_t CGDisplayIOServicePort (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

The I/O Kit service port for the specified display.

Discussion

An I/O Kit service port can be passed to I/O Kit to obtain additional information about the display.

The port is owned by the graphics system, and should not be destroyed.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayIsActive

Returns a Boolean value indicating whether a display is active.

```
boolean_t CGDisplayIsActive (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

If `true`, the specified display is active; otherwise, `false`.

Discussion

An active display is connected, awake, and available for drawing. In a hardware mirroring set, only the primary display is active.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayIsAlwaysInMirrorSet

Returns a Boolean value indicating whether a display is always in a mirroring set.

```
boolean_t CGDisplayIsAlwaysInMirrorSet (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

If `true`, the specified display is in a mirroring set and cannot be removed from this set.

Discussion

Some hardware configurations support the connection of auxiliary displays that always mirror the main display, and therefore cannot be removed from the mirroring set to which they belong.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayIsAsleep

Returns a Boolean value indicating whether a display is sleeping (and is therefore not drawable.)

```
boolean_t CGDisplayIsAsleep (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

If `true`, the specified display is in sleep mode; otherwise, `false`.

Discussion

A display is sleeping when its frame buffer and the attached monitor are in reduced power mode. A sleeping display is still considered to be a part of global display (desktop) space, but it is not drawable.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayIsBuiltin

Returns a Boolean value indicating whether a display is built-in, such as the internal display in portable systems.

```
boolean_t CGDisplayIsBuiltin (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

If `true`, the specified display is considered to be a built-in display; otherwise, `false`.

Discussion

Portable systems typically identify the internal LCD panel as a built-in display.

Note that it is possible and reasonable for a system to have no displays marked as built-in. For example, a portable system running with the lid closed may report no built-in displays.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayIsCaptured

Returns a Boolean value indicating whether a display is captured.

```
boolean_t CGDisplayIsCaptured (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

If `true`, the specified display is captured; otherwise, `false`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayIsInHWMirrorSet

Returns a Boolean value indicating whether a display is in a hardware mirroring set.

```
boolean_t CGDisplayIsInHWMirrorSet (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value
If `true`, the specified display is a member of a hardware mirroring set; otherwise, `false`.

Discussion
When hardware mirroring is enabled, the contents of a single frame buffer are rendered in all displays in the hardware mirroring set. All drawing operations are directed to the primary display in the set—see `CGDisplayPrimaryDisplay` (page 1506).

For more information about display mirroring, see `CGConfigureDisplayMirrorOfDisplay` (page 1481).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayIsInMirrorSet

Returns a Boolean value indicating whether a display is in a mirroring set.

```
boolean_t CGDisplayIsInMirrorSet (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
If `true`, the specified display is a member of a software or hardware mirroring set; otherwise, `false`.

Discussion
For more information about display mirroring, see `CGConfigureDisplayMirrorOfDisplay` (page 1481).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayIsMain

Returns a Boolean value indicating whether a display is the main display.

```
boolean_t CGDisplayIsMain (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
If `true`, the specified display is currently the main display; otherwise, `false`.

Discussion
For information about the characteristics of a main display, see `CGMainDisplayID` (page 1522).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayIsOnline

Returns a Boolean value indicating whether a display is connected or online.

```
boolean_t CGDisplayIsOnline (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
If `true`, the specified display is connected; otherwise, `false`.

Discussion
A display is considered connected or online when the frame buffer hardware is connected to a monitor.

You can use this function to determine if someone has hot-plugged a display to the system. Note that hot-plugging is a hardware feature that may not be present on all displays.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayIsStereo

Returns a Boolean value indicating whether a display is running in a stereo graphics mode.

```
boolean_t CGDisplayIsStereo (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
If `true`, the specified display is running in a stereo graphics mode; otherwise, `false`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayMirrorsDisplay

For a secondary display in a mirroring set, returns the primary display.

```
CGDirectDisplayID CGDisplayMirrorsDisplay (
   CGDirectDisplayID display
);
```

Parameters

display
A secondary display in a mirroring set.

Return Value
Returns the primary display in the mirroring set. Returns `kCGNullDirectDisplay` if the specified display is actually the primary display or is not in a mirroring set.

Discussion
For more information about display mirroring, see `CGConfigureDisplayMirrorOfDisplay` (page 1481).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayModelNumber

Returns the model number of a display monitor.

```
uint32_t CGDisplayModelNumber (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
A model number for the monitor associated with the specified display, or a constant to indicate an exception—see the discussion below.

Discussion
This function uses I/O Kit to identify the monitor associated with the specified display. The return value depends on the following:

- If I/O Kit can identify the monitor, the product ID code for the monitor is returned.
- If I/O Kit can't identify the monitor, `kDisplayProductIDGeneric` is returned.
- If no monitor is connected, a value of `0xFFFFFFFF` is returned.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayMoveCursorToPoint

Moves the mouse cursor to a specified point relative to the display origin (the upper left corner of the display).

```
CGDisplayErr CGDisplayMoveCursorToPoint (
   CGDirectDisplayID display,
   CGPoint point
);
```

Parameters

display
The display to access.

point
The coordinates of a point in local display space. The origin is the upper left corner of the specified display.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
No events are generated as a result of this move. Points that would lie outside the desktop are clipped to the desktop.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayPixelsHigh

Returns the display height in pixel units.

```
size_t CGDisplayPixelsHigh (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The display height in pixel units.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayPixelsWide

Returns the display width in pixel units.

```
size_t CGDisplayPixelsWide (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The display width in pixel units.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayPrimaryDisplay

Returns the primary display in a hardware mirroring set.

```
CGDirectDisplayID CGDisplayPrimaryDisplay (
   CGDirectDisplayID display
);
```

Parameters

display
A display in a hardware mirror set.

Return Value
The primary display in the mirror set. If `display` is not hardware-mirrored, this function simply returns `display`.

Discussion
In hardware mirroring, the contents of a single frame buffer are rendered in two or more displays simultaneously. The mirrored displays are said to be in a *hardware mirroring set*.

At the discretion of the device driver, one of the displays in a hardware mirroring set is designated as the *primary* display. The device driver binds the drawing engine, hardware accelerator, and 3D engine to the primary display, and directs all drawing operations to this display.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayRegisterReconfigurationCallback

Registers a callback function to be invoked whenever a local display is reconfigured.

```
CGError CGDisplayRegisterReconfigurationCallback (
   CGDisplayReconfigurationCallBack proc,
   void *userInfo
);
```

Parameters

proc

A pointer to the callback function to be registered.

userInfo

A pointer to user-defined data, or `NULL`. The `userInfo` argument is passed back to the callback function each time it's invoked.

Discussion

Whenever local displays are reconfigured, the callback function you register is invoked twice for each display that's added, removed, or currently online—once before the reconfiguration, and once after the reconfiguration. For more information, see the callback type `CGDisplayReconfigurationCallBack` (page 1540).

A callback function may be registered multiple times with different user-defined data pointers, resulting in multiple registration entries. For each registration, when notification is no longer needed you should remove the registration by calling the function `CGDisplayRemoveReconfigurationCallback` (page 1508).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayRelease

Releases a captured display.

```
CGDisplayErr CGDisplayRelease (
   CGDirectDisplayID display
);
```

Parameters

display

The display to release.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayRemoveReconfigurationCallback

Removes the registration of a callback function that's invoked whenever a local display is reconfigured.

```
CGError CGDisplayRemoveReconfigurationCallback (
   CGDisplayReconfigurationCallBack proc,
   void *userInfo
);
```

Parameters

proc

A pointer to the callback function associated with the registration to be removed.

userInfo

A pointer to user-defined data associated with the registration to be removed, or `NULL`. This is the same pointer that's passed to the function `CGDisplayRegisterReconfigurationCallback` (page 1507) when registering the callback.

Discussion

When you call this function, the two arguments must match the registered entry to be removed.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayRestoreColorSyncSettings

Restores the gamma tables to the values in the user's ColorSync display profile.

```
void CGDisplayRestoreColorSyncSettings (
   void
);
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGDisplayRotation

Returns the rotation angle of a display in degrees.

```
double CGDisplayRotation (
    CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

The rotation angle of the display in degrees, or 0 if the display is not valid.

Discussion
This function returns the rotation angle of a display in a clockwise direction. For example, if the specified display is rotated clockwise 90 degrees then this function returns 90.0. After a 90 degree clockwise rotation, the physical bottom of the display is on the left side and the physical top is on the right side.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplaySamplesPerPixel

Returns the number of color components used to represent a pixel.

```
size_t CGDisplaySamplesPerPixel (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The number of color components used to represent a pixel.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayScreenSize

Returns the width and height of a display in millimeters.

```
CGSize CGDisplayScreenSize (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
The size of the specified display in millimeters, or 0 if the display is not valid.

Discussion
If Extended Display Identification Data (EDID) for the display device is not available, the size is estimated based on the device width and height in pixels from `CGDisplayBounds` (page 1491), with an assumed resolution of 2.835 pixels/mm or 72 DPI, a reasonable guess for displays predating EDID support.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplaySerialNumber

Returns the serial number of a display monitor.

```
uint32_t CGDisplaySerialNumber (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
A serial number for the monitor associated with the specified display, or a constant to indicate an exception—see the discussion below.

Discussion
This function uses I/O Kit to identify the monitor associated with the specified display.

If I/O Kit can identify the monitor:

- If the manufacturer has encoded a serial number for the monitor, the number is returned.
- If there is no encoded serial number, `0x00000000` is returned.

If I/O Kit cannot identify the monitor:

- If a monitor is connected to the display, `0x00000000` is returned.
- If no monitor is connected to the display hardware, a value of `0xFFFFFFFF` is returned.

Note that a serial number is meaningful only in conjunction with a specific vendor and product or model.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplaySetPalette

Sets the palette for a display.

```
CGDisplayErr CGDisplaySetPalette (
   CGDirectDisplayID display,
   const CGDirectPaletteRef palette
);
```

Parameters

display
The display to access.

palette
The display palette to set.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplaySetStereoOperation

Immediately enables or disables stereo operation for a display.

```
CGError CGDisplaySetStereoOperation (
   CGDirectDisplayID display,
   boolean_t stereo,
   boolean_t forceBlueLine,
   CGConfigureOption option
);
```

Parameters

display
The display being configured.

stereo
Pass `true` if you want to enable stereo operation. To disable it, pass `false`.

forceBlueLine
When in stereo operation, a display may need to generate a special stereo sync signal as part of the video output. The sync signal consists of a blue line which occupies the first 25% of the last scanline for the left eye view, and the first 75% of the last scanline for the right eye view. The remainder of the scanline is black. To force the display to generate this sync signal, pass `true`; otherwise pass `false`.

option
A constant that specifies the scope of the display configuration changes. For more information, see "Display Configuration Scopes" (page 1555).

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
The system normally detects the presence of a stereo window and automatically switches a display containing a stereo window to stereo operation. This function provides a mechanism to force a display to stereo operation immediately, and to set options (blue line sync signal) when in stereo operation.

On success, the display resolution, mirroring mode, and available display modes may change due to hardware-specific capabilities and limitations. You should check these settings to verify that they are appropriate for your application.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGDisplayConfiguration.h

CGDisplayShowCursor

Decrements the hide cursor count, and shows the mouse cursor if the count is zero.

```
CGDisplayErr CGDisplayShowCursor (
   CGDirectDisplayID display
);
```

Parameters

display
This parameter is not used. By default, you may pass kCGDirectMainDisplay.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
If the hide cursor count is zero, this function shows the cursor regardless of its current location; the display parameter is ignored. In most cases, the caller must be the foreground application to affect the cursor.

Availability
Available in Mac OS X v10.0 and later.

See Also
CGDisplayHideCursor (page 1497)

Declared In
CGDirectDisplay.h

CGDisplaySwitchToMode

Switches a display to a different mode.

```
CGDisplayErr CGDisplaySwitchToMode (
   CGDirectDisplayID display,
   CFDictionaryRef mode
);
```

Parameters

display
The display to access.

mode
A display mode dictionary that contains information about the display mode to set. The dictionary passed in must be a dictionary returned by another Quartz display function such as CGDisplayAvailableModes (page 1486) or CGDisplayBestModeForParameters (page 1488). For a list of the properties in a display mode dictionary, see "Display Mode Standard Properties" (page 1557) and "Display Mode Optional Properties" (page 1558). For general information about using dictionaries, see *CFDictionary Reference*.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
This function switches the display mode of the specified display. The operation is always synchronous; the function does not return until the mode switch is complete. Note that after switching, display parameters and addresses may change.

The selected display mode persists for the life of the calling program. When the program terminates, the display mode automatically reverts to the permanent setting in the Displays panel of System Preferences.

When changing the display mode of a display in a mirroring set, other displays in the mirroring set will be assigned a mode that's capable of mirroring the bounds of the display being adjusted. To avoid this automatic behavior, you can use the following procedure: call `CGBeginDisplayConfiguration`, call `CGConfigureDisplayMode` for each display to explicitly set the mode, and finally call `CGCompleteDisplayConfiguration`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDisplayUnitNumber

Returns the logical unit number of a display.

```
uint32_t CGDisplayUnitNumber (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
A logical unit number for the specified display.

Discussion
The logical unit number represents a particular node in the I/O Kit device tree associated with the display's frame buffer. For a particular hardware configuration, this value will not change when the attached monitor is changed.

The unit number will change if the I/O Kit device tree changes, as when hardware is reconfigured, drivers are replaced, or significant changes occur to I/O Kit, so it should not be assumed to be invariant across login sessions.

For more information about I/O Kit, see the Apple publication *"I/O Kit Fundamentals"*.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayUsesOpenGLAcceleration

Returns a Boolean value indicating whether Quartz is using OpenGL-based window acceleration (Quartz Extreme) to render in a display.

```
boolean_t CGDisplayUsesOpenGLAcceleration (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

If `true`, Quartz Extreme is used to render in the specified display; otherwise, `false`.

Discussion

Quartz Extreme is an OpenGL-based, hardware-accelerated window compositor available in Mac OS X version 10.2 and later. Quartz Extreme requires a minimum hardware configuration to operate.

The information this function provides is typically used to adjust the demands of drawing operations to the capabilities of the display hardware. For example, an application running on an unaccelerated system could disable live window-resizing.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CGDisplayConfiguration.h`

CGDisplayVendorNumber

Returns the vendor number of the specified display's monitor.

```
uint32_t CGDisplayVendorNumber (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

A vendor number for the monitor associated with the specified display, or a constant to indicate an exception—see the discussion below.

Discussion

This function uses I/O Kit to identify the monitor associated with the specified display.

There are three cases:

- If I/O Kit can identify the monitor, the vendor ID is returned.
- If I/O Kit cannot identify the monitor, `kDisplayVendorIDUnknown` is returned.
- If there is no monitor associated with the display, `0xFFFFFFFF` is returned.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGDisplayWaitForBeamPositionOutsideLines

Waits until the beam position moves outside a region in a display screen. This function is not designed for VBL drawing synchronization.

```
CGDisplayErr CGDisplayWaitForBeamPositionOutsideLines (
   CGDirectDisplayID display,
   CGBeamPosition upperScanLine,
   CGBeamPosition lowerScanLine
);
```

Parameters

display
The display to access.

upperScanLine
The upper scan line number.

lowerScanLine
The lower scan line number.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
This function waits until the beam position is outside the range specified by the arguments `upperScanLine` and `lowerScanLine`. If the value of `upperScanLine` is greater than the value of `lowerScanLine`, or if `upperScanLine` and `lowerScanLine` encompass the entire display height, this function returns an error.

Some displays may not use conventional video vertical and horizontal sweep in painting. These displays report a `kCGDisplayRefreshRate` of 0 in the dictionary returned by `CGDisplayCurrentMode` (page 1494). Also, some display device drivers may not implement support for this mechanism. On such displays, this function returns at once.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGGetActiveDisplayList

Provides a list of displays that are active (or drawable).

```
CGDisplayErr CGGetActiveDisplayList (
   CGDisplayCount maxDisplays,
   CGDirectDisplayID *activeDspys,
   CGDisplayCount *dspyCnt
);
```

Parameters

maxDisplays

The size of the `activeDspys` array. This value determines the maximum number of displays that can be returned.

activeDspys

A pointer to storage provided by the caller for an array of display IDs. On return, the array contains a list of active displays. If you pass `NULL`, on return the display count contains the total number of active displays.

dspyCnt

A pointer to a display count variable provided by the caller. On return, the display count contains the actual number of displays returned in the `activeDspys` array. This value is at most `maxDisplays`.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
The first entry in the list of active displays is the main display. In case of mirroring, the first entry is the largest drawable display or, if all are the same size, the display with the greatest pixel depth.

Note that when hardware mirroring is being used between displays, only the primary display is active and appears in the list. When software mirroring is being used, all the mirrored displays are active and appear in the list. For more information about mirroring, see `CGConfigureDisplayMirrorOfDisplay` (page 1481).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`CGDirectDisplay.h`

CGGetDisplaysWithOpenGLDisplayMask

Provides a list of displays that corresponds to the bits set in an OpenGL display mask.

```
CGDisplayErr CGGetDisplaysWithOpenGLDisplayMask (
   CGOpenGLDisplayMask mask,
   CGDisplayCount maxDisplays,
   CGDirectDisplayID *dspys,
   CGDisplayCount *dspyCnt
);
```

Parameters

mask

An OpenGL display mask that identifies one or more displays.

maxDisplays

The size of the `dspys` array. This value determines the maximum number of displays that can be returned.

dspys

A pointer to storage provided by the caller for an array of display IDs. On return, the array contains a list of displays that corresponds to the bits set in the mask. If you pass `NULL`, on return the display count contains the total number of displays specified in the mask.

dspyCnt

A pointer to a display count variable provided by the caller. On return, the display count contains the actual number of displays returned in the `dspys` array. This value is at most `maxDisplays`.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGGetDisplaysWithPoint

Provides a list of online displays with bounds that include the specified point.

```
CGDisplayErr CGGetDisplaysWithPoint (
   CGPoint point,
   CGDisplayCount maxDisplays,
   CGDirectDisplayID *dspys,
   CGDisplayCount *dspyCnt
);
```

Parameters

point

The coordinates of a point in global display space. The origin is the upper left corner of the main display.

maxDisplays

The size of the `dspys` array. This value determines the maximum number of displays that can be returned.

dspys

A pointer to storage provided by the caller for an array of display IDs. On return, the array contains a list of displays with bounds that include the point. If you pass `NULL`, on return the display count contains the total number of displays with bounds that include the point.

dspyCnt

A pointer to a display count variable provided by the caller. On return, the display count contains the actual number of displays returned in the `dspys` array. This value is at most `maxDisplays`.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGGetDisplaysWithRect

Gets a list of online displays with bounds that intersect the specified rectangle.

```
CGDisplayErr CGGetDisplaysWithRect (
   CGRect rect,
   CGDisplayCount maxDisplays,
   CGDirectDisplayID *dspys,
   CGDisplayCount *dspyCnt
);
```

Parameters

rect

The location and size of a rectangle in global display space. The origin is the upper left corner of the main display.

maxDisplays

The size of the *dspys* array. This value determines the maximum number of displays that can be returned in the *dspys* parameter. Generally, you should specify a number greater than 0 for this parameter. If you specify 0, the value returned in *dspyCnt* is undefined and this function sets the *dspys* parameter to NULL.

dspys

A pointer to storage provided by the caller for an array of display IDs. On return, the array contains a list of displays whose bounds intersect the specified rectangle.

dspyCnt

A pointer to a display count variable provided by the caller. On return, this variable contains the number of displays that were returned in the *dspys* parameter. You must provide a non-NULL value for this parameter.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGGetDisplayTransferByFormula

Gets the coefficients of the gamma transfer formula for a display.

```
CGDisplayErr CGGetDisplayTransferByFormula (
   CGDirectDisplayID display,
   CGGammaValue *redMin,
   CGGammaValue *redMax,
   CGGammaValue *redGamma,
   CGGammaValue *greenMin,
   CGGammaValue *greenMax,
   CGGammaValue *greenGamma,
   CGGammaValue *blueMin,
   CGGammaValue *blueMax,
   CGGammaValue *blueGamma
);
```

Parameters

display
The display to access.

redMin
The minimum value of the red channel in the gamma table. The value is a number in the interval [0, redMax).

redMax
The maximum value of the red channel in the gamma table. The value is a number in the interval (redMin, 1].

redGamma
A positive value used to compute the red channel in the gamma table.

greenMin
The minimum value of the green channel in the gamma table. The value is a number in the interval [0, greenMax).

greenMax
The maximum value of the green channel in the gamma table. The value is a number in the interval (greenMin, 1].

greenGamma
A positive value used to compute the green channel in the gamma table.

blueMin
The minimum value of the blue channel in the gamma table. The value is a number in the interval [0, blueMax).

blueMax
The maximum value of the blue channel in the gamma table. The value is a number in the interval (blueMin, 1].

blueGamma
A positive value used to compute the blue channel in the gamma table.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
For information about the gamma transfer formula, see the description of the function `CGSetDisplayTransferByFormula` (page 1533).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGGetDisplayTransferByTable

Gets the values in the RGB gamma tables for a display.

```
CGDisplayErr CGGetDisplayTransferByTable (
   CGDirectDisplayID display,
   CGTableCount capacity,
   CGGammaValue *redTable,
   CGGammaValue *greenTable,
   CGGammaValue *blueTable,
   CGTableCount *sampleCount
);
```

Parameters

display
The display to access.

capacity
The number of entries each table can hold.

redTable
A pointer to an array of type `CGGammaValue` with size `capacity`. On return, the array contains the values of the red channel in the display's gamma table.

greenTable
A pointer to an array of type `CGGammaValue` with size `capacity`. On return, the array contains the values of the green channel in the display's gamma table.

blueTable
A pointer to an array of type `CGGammaValue` with size `capacity`. On return, the array contains the values of the blue channel in the display's gamma table.

sampleCount
The number of samples actually copied into each array.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGGetLastMouseDelta

Reports the change in mouse position since the last mouse movement event received by the application.

```
void CGGetLastMouseDelta (
   CGMouseDelta *deltaX,
   CGMouseDelta *deltaY
);
```

Parameters

deltaX

A pointer to a `CGMouseDelta` variable. On return, this variable contains the horizontal change in the mouse position since the last mouse movement event.

deltaY

A pointer to a `CGMouseDelta` variable. On return, this variable contains the vertical change in the mouse position since the last mouse movement event.

Discussion

This function is not recommended for general use. Instead, you should use the mouse tracking functions in the Carbon Event Manager.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGGetOnlineDisplayList

Provides a list of displays that are online (active, mirrored, or sleeping).

```
CGDisplayErr CGGetOnlineDisplayList (
   CGDisplayCount maxDisplays,
   CGDirectDisplayID *onlineDspys,
   CGDisplayCount *dspyCnt
);
```

Parameters

maxDisplays

The size of the `onlineDspys` array. This value determines the maximum number of display IDs that can be returned.

onlineDspys

A pointer to storage provided by the caller for an array of display IDs. On return, the array contains a list of the online displays. If you pass `NULL`, on return the display count contains the total number of online displays.

dspyCnt

A pointer to a display count variable provided by the caller. On return, the display count contains the actual number of displays returned in the `onlineDspys` array. This value is at most `maxDisplays`.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

If the frame buffer hardware is connected, a display is considered connected or online.

When hardware mirroring is used, a display can be online but not active or drawable. Programs which manipulate display settings such as the palette or gamma tables need access to all displays, including hardware mirrors which are not drawable.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDirectDisplay.h`

CGMainDisplayID

Returns the display ID of the main display.

```
CGDirectDisplayID CGMainDisplayID (
   void
);
```

Return Value
The display ID assigned to the main display.

Discussion
The main display is the display with its screen location at (0,0) in global coordinates. In a system without display mirroring, the display with the menu bar is typically the main display.

If mirroring is enabled and the menu bar appears on more than one display, this function provides a reliable way to find the main display.

In case of hardware mirroring, the drawable display becomes the main display. In case of software mirroring, the display with the highest resolution and deepest pixel depth typically becomes the main display.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
LiveVideoMixer2

Declared In
`CGDirectDisplay.h`

CGOpenGLDisplayMaskToDisplayID

Maps an OpenGL display mask to a display ID.

```
CGDirectDisplayID CGOpenGLDisplayMaskToDisplayID (
   CGOpenGLDisplayMask mask
);
```

Parameters

mask
The OpenGL display mask to be converted.

Return Value
The display ID assigned to the specified display mask, or `kCGNullDirectDisplay` if no display matches the mask.

Discussion
OpenGL sometimes identifies a display using a bitmask with one bit set. This function maps such a display mask to the corresponding display ID. If you pass in a mask with multiple bits set, this function returns a display ID matching one of these bits.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDirectDisplay.h`

CGPaletteCreateCopy

Returns a copy of a specified display palette.

```
CGDirectPaletteRef CGPaletteCreateCopy (
   CGDirectPaletteRef palette
);
```

Parameters

palette
The display palette to copy.

Return Value
A new display palette object. When you no longer need the palette, you should release it using the function `CGPaletteRelease` (page 1528).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectPalette.h`

CGPaletteCreateDefaultColorPalette

Returns a new display palette representing the default 8-bit color palette.

```
CGDirectPaletteRef CGPaletteCreateDefaultColorPalette (
   void
);
```

Return Value
A new display palette object. When you no longer need the palette, you should release it using the function `CGPaletteRelease` (page 1528).

Discussion
Palettes are used with 256 color display modes. The default palette is the old default 8-bit Mac OS palette, with white at index 0 and black at index 255.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectPalette.h`

CGPaletteCreateFromPaletteBlendedWithColor

Returns a new tinted display palette. The new palette is derived from an existing palette blended with a solid color, at a specified level of intensity.

```
CGDirectPaletteRef CGPaletteCreateFromPaletteBlendedWithColor (
   CGDirectPaletteRef palette,
   CGPaletteBlendFraction fraction,
   CGDeviceColor color
);
```

Parameters

palette

The palette to blend.

fraction

A value between 0 and 1 that represents the blend intensity. See `CGPaletteBlendFraction` (page 1551).

color

The blend color. See `CGDeviceColor` (page 1545).

Return Value

A new display palette object. When you no longer need the palette, you should release it using the function `CGPaletteRelease` (page 1528).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectPalette.h`

CGPaletteCreateWithByteSamples

Returns a new display palette using 8-bit sample data.

```
CGDirectPaletteRef CGPaletteCreateWithByteSamples (
   CGDeviceByteColor *sampleTable,
   CGTableCount sampleCount
);
```

Parameters

sampleTable

A color table with integer values that represent the intensity of the red,green, and blue components in each table entry. Each value ranges from 0 (no color) to 255 (full intensity). See `CGDeviceByteColor` (page 1544).

sampleCount

The number of entries in the specified color table.

Return Value

A new display palette object. When you no longer need the palette, you should release it using the function `CGPaletteRelease` (page 1528).

Availability

Available in Mac OS X v10.0 and later.

Declared In
CGDirectPalette.h

CGPaletteCreateWithCapacity

Returns a new display palette with a specified capacity. The new palette is initialized from the default color palette.

```
CGDirectPaletteRef CGPaletteCreateWithCapacity (
   CGTableCount capacity
);
```

Parameters

capacity
The number of entries in the new palette.

Return Value
A new display palette object. When you no longer need the palette, you should release it using the function CGPaletteRelease (page 1528).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectPalette.h

CGPaletteCreateWithDisplay

Returns a copy of the current palette for a display.

```
CGDirectPaletteRef CGPaletteCreateWithDisplay (
   CGDirectDisplayID display
);
```

Parameters

display
The display to access.

Return Value
A new display palette object, or NULL if the current display mode does not support a palette. When you no longer need the palette, you should release it using the function CGPaletteRelease (page 1528).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectPalette.h

CGPaletteCreateWithSamples

Returns a new display palette using RGB sample data.

```
CGDirectPaletteRef CGPaletteCreateWithSamples (
   CGDeviceColor *sampleTable,
   CGTableCount sampleCount
);
```

Parameters

sampleTable

A color table with floating point values that represent the intensity of the red,green, and blue components in each table entry. Each value ranges from 0 (no color) to 1 (full intensity). See `CGDeviceColor` (page 1545).

sampleCount

The number of entries in the specified color table.

Return Value

A new display palette object. When you no longer need the palette, you should release it using the function `CGPaletteRelease` (page 1528).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectPalette.h`

CGPaletteGetColorAtIndex

Returns the color value at the specified index.

```
CGDeviceColor CGPaletteGetColorAtIndex (
   CGDirectPaletteRef palette,
   CGTableCount index
);
```

Parameters

palette

The display palette to access.

index

The zero-based index of the desired palette entry.

Return Value

A color value. See `CGDeviceColor` (page 1545).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectPalette.h`

CGPaletteGetIndexForColor

Returns the index of the display palette entry that most closely matches a specified color value.

```
CGTableCount CGPaletteGetIndexForColor (
   CGDirectPaletteRef palette,
   CGDeviceColor color
);
```

Parameters

palette

The display palette to access.

color

The color value to match. See `CGDeviceColor` (page 1545).

Return Value

The index of the display palette entry that most closely matches the specified color value.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectPalette.h`

CGPaletteGetNumberOfSamples

Returns the number of colors in a display palette.

```
CGTableCount CGPaletteGetNumberOfSamples (
   CGDirectPaletteRef palette
);
```

Parameters

palette

The display palette to access.

Return Value

The number of colors in the specified display palette.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectPalette.h`

CGPaletteIsEqualToPalette

Returns a Boolean value indicating whether two display palettes are equal.

```
Boolean CGPaletteIsEqualToPalette (
   CGDirectPaletteRef palette1,
   CGDirectPaletteRef palette2
);
```

Parameters

palette1

The first display palette to compare.

palette2
The second display palette to compare.

Return Value
If `true`, the two specified display palettes are equal; otherwise, `false`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectPalette.h`

CGPaletteRelease

Decrements the retain count of a display palette.

```
void CGPaletteRelease (
   CGDirectPaletteRef palette
);
```

Parameters

palette
The display palette to release.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectPalette.h`

CGPaletteSetColorAtIndex

Updates the color value at the specified index in a display palette.

```
void CGPaletteSetColorAtIndex (
   CGDirectPaletteRef palette,
   CGDeviceColor color,
   CGTableCount index
);
```

Parameters

palette
The display palette to access.

color
The new color value.

index
The index of the palette entry to update.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectPalette.h`

CGRegisterScreenRefreshCallback

Registers a callback function to be invoked when local displays are refreshed or modified.

```
CGError CGRegisterScreenRefreshCallback (
   CGScreenRefreshCallback function,
   void *userParameter
);
```

Parameters

function

A pointer to the callback function to be registered.

userParameter

A pointer to user-defined data, or `NULL`. The `userParameter` argument is passed back to the callback function each time it's invoked.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

A callback function may be registered multiple times with different user-defined data pointers, resulting in multiple registration entries. For each registration, when notification is no longer needed you should call the function `CGUnregisterScreenRefreshCallback` (page 1536) to remove the registration.

The callback function you register is invoked only if your application has an active event loop. The callback is invoked in the same thread of execution that is processing events within your application.

Special Considerations

In Mac OS X v10.4 and earlier, the result code returned by this function is a random value and should be ignored. In Mac OS X v10.5 and later, the result code is valid.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGReleaseAllDisplays

Releases all captured displays.

```
CGDisplayErr CGReleaseAllDisplays (
   void
);
```

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

This function releases all captured displays and restores the display modes to the user's preferences. It may be used in conjunction with any of the functions that capture displays, such as `CGCaptureAllDisplays` (page 1478).

Availability

Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGReleaseDisplayFadeReservation

Releases a display fade reservation, and unfades the display if needed.

```
CGError CGReleaseDisplayFadeReservation (
   CGDisplayFadeReservationToken myToken
);
```

Parameters

myToken

The current fade reservation token to be released. On return, the reservation token is no longer valid and should be discarded.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
If you call this function while an asynchronous fade operation is running, there are two possible outcomes:

- If the ending blend value is `kCGDisplayBlendNormal`, the fade operation is allowed to run to completion.
- If the ending blend value is not `kCGDisplayBlendNormal`, the fade operation is terminated immediately and the display is returned to normal.

In both cases, the reservation is actually released when the fade operation completes.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGDisplayFade.h

CGReleaseScreenRefreshRects

Deallocates a list of rectangles that represent changed areas on local displays.

```
void CGReleaseScreenRefreshRects (
   CGRect *rectArray
);
```

Parameters

rectArray

A list of rectangles obtained by calling `CGWaitForScreenRefreshRects` (page 1537) or `CGWaitForScreenUpdateRects` (page 1538).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGRemoteOperation.h

CGRestorePermanentDisplayConfiguration

Restores the permanent display configuration settings for the current user.

```
void CGRestorePermanentDisplayConfiguration (
   void
);
```

Discussion
This function provides a convenient way to restore the permanent display configuration.

Applications that temporarily change the display configuration—such as applications and games that switch to full-screen display mode—can use this function to undo the changes.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayConfiguration.h`

CGScreenRegisterMoveCallback

Registers a callback function to be invoked when an area of the display is moved.

```
CGError CGScreenRegisterMoveCallback (
   CGScreenUpdateMoveCallback function,
   void *userParameter
);
```

Parameters

function
A pointer to the callback function to be registered.

userParameter
A pointer to user-defined data, or `NULL`. The `userParameter` argument is passed back to the callback function each time it's invoked.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
A callback function may be registered multiple times with different user-defined data pointers, resulting in multiple registration entries. For each registration, when notification is no longer needed you should remove the registration by calling the function `CGScreenUnregisterMoveCallback` (page 1532).

The callback function you register is invoked only if your application has an active event loop. The callback is invoked in the same thread of execution that is processing events within your application.

Special Considerations

This function is implemented in Mac OS X version 10.4.3 and later.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGRemoteOperation.h`

CGScreenUnregisterMoveCallback

Removes a previously registered callback function invoked when an area of the display is moved.

```
void CGScreenUnregisterMoveCallback (
   CGScreenUpdateMoveCallback function,
   void *userParameter
);
```

Parameters

function

A pointer to the callback function to be unregistered.

userParameter

A pointer to user-defined data, or `NULL`. You should pass the same value you used when you registered the callback function.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

When you call this function, the two arguments must match the registered entry to be removed.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGRemoteOperation.h`

CGSessionCopyCurrentDictionary

Returns information about the caller's window server session.

```
CFDictionaryRef CGSessionCopyCurrentDictionary (
   void
);
```

Return Value

A window server session dictionary, or `NULL` if the caller is not running within a Quartz GUI session or the window server is disabled. You should release the dictionary when you are finished using it. For information about the key-value pairs in this dictionary, see "Window Server Session Properties" (page 1563).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CGSession.h`

CGSetDisplayTransferByByteTable

Sets the byte values in the 8-bit RGB gamma tables for a display.

```
CGDisplayErr CGSetDisplayTransferByByteTable (
   CGDirectDisplayID display,
   CGTableCount tableSize,
   const CGByteValue *redTable,
   const CGByteValue *greenTable,
   const CGByteValue *blueTable
);
```

Parameters

display

The display to access.

tableSize

The number of entries in each table.

redTable

An array of size `tableSize` containing the byte values of the red channel in the display's gamma table.

greenTable

An array of size `tableSize` containing the byte values of the green channel in the display's gamma table.

blueTable

An array of size `tableSize` containing the byte values of the blue channel in the display's gamma table.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

The same table may be passed in for the red, green, and blue channels. The tables are interpolated as needed to generate the number of samples required by the graphics hardware.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGSetDisplayTransferByFormula

Sets the gamma function for a display, by specifying the coefficients of the gamma transfer formula.

```
CGDisplayErr CGSetDisplayTransferByFormula (
   CGDirectDisplayID display,
   CGGammaValue redMin,
   CGGammaValue redMax,
   CGGammaValue redGamma,
   CGGammaValue greenMin,
   CGGammaValue greenMax,
   CGGammaValue greenGamma,
   CGGammaValue blueMin,
   CGGammaValue blueMax,
   CGGammaValue blueGamma
);
```

Parameters

display

The display to access.

redMin

The minimum value of the red channel in the gamma table. The value should be a number in the interval [0, redMax).

redMax

The maximum value of the red channel in the gamma table. The value should be a number in the interval (redMin, 1].

redGamma

A positive value used to compute the red channel in the gamma table.

greenMin

The minimum value of the green channel in the gamma table. The value should be a number in the interval [0, greenMax).

greenMax

The maximum value of the green channel in the gamma table. The value should be a number in the interval (greenMin, 1].

greenGamma

A positive value used to compute the green channel in the gamma table.

blueMin

The minimum value of the blue channel in the gamma table. The value should be a number in the interval [0, blueMax).

blueMax

The maximum value of the blue channel in the gamma table. The value should be a number in the interval (blueMin, 1].

blueGamma

A positive value used to compute the blue channel in the gamma table.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

This function uses the specified parameter values to compute a gamma correction table for the specified display. The values in the table are computed by sampling the following gamma transfer formula for a range of indices from 0 to 1:

```
value = Min + ((Max - Min) * pow(index, Gamma))
```

The resulting values are converted to a machine-specific format and loaded into display hardware.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGSetDisplayTransferByTable

Sets the color gamma function for a display, by specifying the values in the RGB gamma tables.

```
CGDisplayErr CGSetDisplayTransferByTable (
   CGDirectDisplayID display,
   CGTableCount tableSize,
   const CGGammaValue *redTable,
   const CGGammaValue *greenTable,
   const CGGammaValue *blueTable
);
```

Parameters

display
The display to access.

tableSize
The number of entries in each table.

redTable
An array of size `tableSize` containing the values of the red channel in the display's gamma table. The values should be in the range 0.0 to 1.0.

greenTable
An array of size `tableSize` containing the values of the green channel in the display's gamma table. The values should be in the range 0.0 to 1.0.

blueTable
An array of size `tableSize` containing the values of the blue channel in the display's gamma table. The values should be in the range 0.0 to 1.0.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
The same table may be passed in for the red, green, and blue channels. The tables are interpolated as needed to generate the number of samples required by the graphics hardware.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGShieldingWindowID

Returns the window ID of the shield window for a captured display.

```
uint32_t CGShieldingWindowID (
   CGDirectDisplayID display
);
```

Parameters

display

The display to access.

Return Value

The window ID of the shield window for the specified display, or `NULL` if the display is not shielded.

Discussion

To prevent updates by direct-to-screen programs (such as Classic), Quartz draws a shield window that fills the entire screen of a captured display.

This function is not recommended for use in applications. Note that the graphics context associated with this window is not a full-featured drawing context. To get a full-featured drawing context for a captured display, you should use the function `CGDisplayGetDrawingContext` (page 1497).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGShieldingWindowLevel

Returns the window level of the shield window for a captured display.

```
int32_t CGShieldingWindowLevel (
   void
);
```

Return Value

The window level of the shield window for a captured display.

Discussion

This function returns a value that is sometimes used to position a window over the shield window for a captured display. Attempting to position a window over a captured display may be unsuccessful—or may present undesirable results such as illegible or invisible content—because of interactions between full-screen graphics (such as OpenGL full-screen drawing contexts) and the graphics hardware. Because of these limitations, and because the implementation of display capture may change in the future, this technique is not recommended.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGDirectDisplay.h`

CGUnregisterScreenRefreshCallback

Removes a previously registered callback function invoked when local displays are refreshed or modified.

```
void CGUnregisterScreenRefreshCallback (
   CGScreenRefreshCallback function,
   void *userParameter
);
```

Parameters

function

A pointer to the callback function to be unregistered.

userParameter

A pointer to user-defined data, or `NULL`. You should pass the same value you used when you registered the callback function.

Discussion

When you call this function, the two arguments must match the registered entry to be removed.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGWaitForScreenRefreshRects

Waits for screen refresh operations.

```
CGError CGWaitForScreenRefreshRects (
   CGRect **pRectArray,
   CGRectCount *pCount
);
```

Parameters

pRectArray

A pointer to a `CGRect*` variable. On return, the variable contains an array of rectangles that bound the refreshed areas, specified in global coordinates. When you no longer need the array, you should deallocate it by calling `CGReleaseScreenRefreshRects` (page 1530).

pCount

A pointer to a `CGRectCount` variable. On return, the variable contains the number of entries in the returned array of rectangles.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

In some applications it may be preferable to wait for screen refresh data synchronously, using this function. You should call this function in a thread other than the main event-processing thread.

As an alternative, Quartz also supports asynchronous notification—see `CGRegisterScreenRefreshCallback` (page 1529). If refresh callback functions are registered, this function should not be used.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGWaitForScreenUpdateRects

Waits for screen update operations.

```
CGError CGWaitForScreenUpdateRects (
   CGScreenUpdateOperation requestedOperations,
   CGScreenUpdateOperation *currentOperation,
   CGRect **pRectArray,
   size_t *pCount,
   CGScreenUpdateMoveDelta *pDelta
);
```

Parameters

requestedOperations

The desired types of screen update operations. There are several possible choices:

- Specify `kCGScreenUpdateOperationRefresh` if you want all move operations to be returned as refresh operations.
- Specify `(kCGScreenUpdateOperationRefresh | kCGScreenUpdateOperationMove)` if you want to distinguish between move and refresh operations.
- Add `kCGScreenUpdateOperationReducedDirtyRectangleCount` to the screen operations if you want to minimize the number of rectangles returned to represent changed areas of the display.

currentOperation

A pointer to a `CGScreenUpdateOperation` variable. On return, the variable indicates the type of update operation (refresh or move).

pRectArray

A pointer to a `CGRect*` variable. On return, the variable contains an array of rectangles that bound the updated areas, specified in global coordinates. When you no longer need the array, you should deallocate it by calling `CGReleaseScreenRefreshRects` (page 1530).

pCount

A pointer to a `size_t` variable. On return, the variable contains the number of entries in the returned array of rectangles.

pDelta

A pointer to a `CGScreenUpdateMoveDelta` variable. On return, if the value of the `currentOperation` parameter is `kCGScreenUpdateOperationMove` the variable contains the distance moved.

Return Value

A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion

In some applications it may be preferable to wait for screen update data synchronously, using this function. You should call this function in a thread other than the main event-processing thread.

As an alternative, Quartz also supports asynchronous notification—see `CGRegisterScreenRefreshCallback` (page 1529) and `CGScreenRegisterMoveCallback` (page 1531). If refresh or move callback functions are registered, this function should not be used.

Special Considerations

This function is implemented in Mac OS X version 10.4.3 and later.

Availability

Available in Mac OS X v10.3 and later.

Declared In
CGRemoteOperation.h

CGWarpMouseCursorPosition

Moves the mouse cursor without generating events.

```
CGError CGWarpMouseCursorPosition (
   CGPoint newCursorPosition
);
```

Parameters

newCursorPosition
The new mouse cursor position in global display coordinates.

Return Value
A result code. See "Quartz Display Services Result Codes" (page 1564).

Discussion
You can use this function to 'warp' or alter the cursor position without generating or posting an event. For example, this function is often used to move the cursor position back to the center of the screen by games that do not want the cursor pinned by display edges.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGRemoteOperation.h

CGWindowLevelForKey

Returns the window level that corresponds to one of the standard window types.

```
CGWindowLevel CGWindowLevelForKey (
   CGWindowLevelKey key
);
```

Parameters

key
A window level key constant that represents one of the standard window types. See "Window Level Keys" (page 1560).

Return Value
The window level that corresponds to the specified key.

Discussion
This function is not recommended for use in applications. (This function is provided for application frameworks that create and manage windows, such as Carbon and Cocoa.)

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGWindowLevel.h

CGWindowServerCFMachPort

Returns a Core Foundation mach port (CFMachPort) that corresponds to the Mac OS X window server.

```
CFMachPortRef CGWindowServerCFMachPort (
   void
);
```

Return Value
A Core Foundation mach port, or `NULL` if the window server is not running. When you no longer need the port, you should release it using the function `CFRelease`.

Discussion
You can use this function to detect if the window server process exits or is not running. If this function returns `NULL`, the window server is not running. This code example shows how to register a callback function to detect when the window server exits:

```
static void handleWindowServerDeath( CFMachPortRef port, void *info  )
{
    printf( "Window Server port death detected!\n" );
    CFRelease(port);
    exit(1);
}

static void watchForWindowServerDeath()
{
    CFMachPortRef port = CGWindowServerCFMachPort();
    CFMachPortSetInvalidationCallBack(port, handleWindowServerDeath);
}
```

Note that this callback will not work unless your program has an active run loop.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CGRemoteOperation.h`

Callbacks

CGDisplayReconfigurationCallBack

A client-supplied callback function that's invoked whenever the configuration of a local display is changed.

```
typedef void (*CGDisplayReconfigurationCallBack) (
   CGDirectDisplayID display,
   CGDisplayChangeSummaryFlags flags,
   void *userInfo
);
```

If you name your function `MyDisplayReconfigurationCallBack`, you would declare it like this:

```
void MyDisplayReconfigurationCallBack (
   CGDirectDisplayID display,
```

```
    CGDisplayChangeSummaryFlags flags,
    void *userInfo
);
```

Parameters

display

The display being reconfigured.

flags

Flags that indicate which display configuration parameters are changing.

userInfo

The `userInfo` argument passed to the function `CGDisplayRegisterReconfigurationCallback` (page 1507) when the callback function is registered.

Discussion

To register a display reconfiguration callback function, you call the function `CGDisplayRegisterReconfigurationCallback` (page 1507). Quartz invokes your callback function when:

- Your application calls a function to reconfigure a local display.
- Your application is listening for events in the event-processing thread, and another application calls a function to reconfigure a local display.
- The user changes the display hardware configuration—for example, by disconnecting a display or changing a system preferences setting.

Before display reconfiguration, Quartz invokes your callback function once for each online display to indicate a pending configuration change. The `flags` argument is always set to `kCGDisplayBeginConfigurationFlag`. Other than the display ID, this callback does not carry other per-display information, as details of how a reconfiguration affects a particular device rely on device-specific behaviors which may not be exposed by a device driver.

After display reconfiguration, Quartz invokes your callback function once for each added, removed, and online display. At this time, all display state reported by Core Graphics, QuickDraw, and the Carbon Display Manager will be up to date. This callback runs after the Carbon Display Manager notification callbacks. The `flags` argument indicates how the display configuration has changed. Note that in the case of removed displays, calls into Quartz with the removed display ID will fail.

The following code example illustrates how to test for specific conditions:

```
void MyDisplayReconfigurationCallBack (
    CGDirectDisplayID display,
    CGDisplayChangeSummaryFlags flags,
    void *userInfo)
{
     if (flags & kCGDisplayAddFlag) {
         // display has been added
     }
     else if (flags & kCGDisplayRemoveFlag) {
         // display has been removed
     }
}
```

Your callback function should avoid attempting to change display configurations, and should not raise exceptions or perform a non-local return such as calling `longjmp`. When you are finished using a callback registration, you should call the function `CGDisplayRemoveReconfigurationCallback` (page 1508) to remove it.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGDisplayConfiguration.h`

CGScreenRefreshCallback

A client-supplied callback function that's invoked when an area of the display is modified or refreshed.

```
typedef void (*CGScreenRefreshCallback) (
   CGRectCount count,
   const CGRect * rectArray,
   void * userParameter
);
```

If you name your function `MyScreenRefreshCallback`, you would declare it like this:

```
void MyScreenRefreshCallback (
   CGRectCount count,
   const CGRect * rectArray,
   void * userParameter
);
```

Parameters

count
The number of rectangles in the `rectArray` parameter.

rectArray
A list of the rectangles in the refreshed areas, specified in global coordinates. You should not modify or deallocate memory pointed to by `rectArray`.

userParameter
The user data you specify when you register this callback.

Discussion
To register a screen refresh callback function, you call the function `CGRegisterScreenRefreshCallback` (page 1529). Quartz invokes your callback function when operations such as drawing, window movement, scrolling, or display reconfiguration occur on local displays. When you are finished using a callback registration, you should call the function `CGUnregisterScreenRefreshCallback` (page 1536) to remove it.

Note that a single rectangle may occupy multiple displays, either by overlapping the displays or by residing on coincident displays when mirroring is active. You can use the function `CGGetDisplaysWithRect` (page 1518) to determine the displays a rectangle occupies.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGRemoteOperation.h

CGScreenUpdateMoveCallback

A client-supplied callback function that's invoked when an area of the display is moved.

```
typedef void (*CGScreenUpdateMoveCallback) (
   CGScreenUpdateMoveDelta delta,
   CGRectCount count,
   const CGRect * rectArray,
   void * userParameter
);
```

If you name your function `MyScreenUpdateMoveCallback`, you would declare it like this:

```
void MyScreenUpdateMoveCallback (
   CGScreenUpdateMoveDelta delta,
   CGRectCount count,
   const CGRect * rectArray,
   void * userParameter
);
```

Parameters

delta
The distance the display area has moved.

count
The number of rectangles in the `rectArray` parameter.

rectArray
A list of the rectangles in the moved areas, specified in global coordinates. The rectangles describe the area prior to the move operation. You should not modify or deallocate memory pointed to by `rectArray`.

userParameter
The user data you specify when you register this callback.

Discussion
To register a screen move callback function, you call the function `CGScreenRegisterMoveCallback` (page 1531). Quartz invokes your callback function when operations such as window movement or scrolling occur on local displays. When you are finished using a callback registration, you should call the function `CGScreenUnregisterMoveCallback` (page 1532) to remove it.

Note that a single rectangle may occupy multiple displays, either by overlapping the displays or by residing on coincident displays when mirroring is active. You can use the function `CGGetDisplaysWithRect` (page 1518) to determine the displays a rectangle occupies.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGRemoteOperation.h

Data Types

CGBeamPosition

Represents a horizontal scan line on a monitor that uses a scanning electron beam to refresh the screen.

```
typedef uint32_t CGBeamPosition;
```

Discussion
CRT and analog-driven displays use a horizontal scanning beam to refresh the screen. The beam position is a number assigned to a horizontal scan line on the screen. Scan lines are numbered `0` to `n-1` from top of screen, where `n` represents the total number of scan lines.

The concept of beam position does not apply to flat-panel LCD displays. While all displays have some concept of scan lines with respect to the frame buffer, LCD displays may not use linear scanning to refresh the screen.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGByteValue

Represents a unit of information in a byte-addressable array or data structure.

```
typedef uint8_t CGByteValue;
```

Discussion
Quartz uses `CGByteValue` to represent integer-based color values in a display palette or a gamma table. For example, see `CGDeviceByteColor` (page 1544).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGDeviceByteColor

Represents a color in a Quartz display palette, using 8-bit integer components.

```
struct CGDeviceByteColor {
    CGByteValue red;
    CGByteValue green;
    CGByteValue blue;
};
typedef struct CGDeviceByteColor CGDeviceByteColor;
```

Fields
`red`
The red component of a palette entry.

`green`
The green component of a palette entry.

`blue`
The blue component of a palette entry.

Discussion
This data structure consists of three integer values that represent the intensity of the red,green, and blue components in a display palette entry. Each component ranges from 0 (no color) to 255 (full intensity).

Quartz provides `CGDeviceByteColor` to allow you to create a display palette using integer-based sample data. Once loaded, you can retrieve color data from the palette only as entries of type `CGDeviceColor` (page 1545).

For more information about display palettes, see `CGDirectPaletteRef` (page 1546).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectPalette.h`

CGDeviceColor

Represents a color in a Quartz display palette.

```
struct CGDeviceColor {
    float red;
    float green;
    float blue;
};
typedef struct CGDeviceColor CGDeviceColor;
```

Fields
`red`
The red component of a palette entry.

`green`
The green component of a palette entry.

`blue`
The blue component of a palette entry.

Discussion
This data structure consists of three floating point values that represent the intensity of the red,green, and blue components in a display palette entry. Each component ranges from 0 (no color) to 1 (full intensity). Values outside this range are clamped to 0 or 1 when the palette is created.

Quartz uses `CGDeviceColor` as the canonical form for a color entry in a display palette. Palette entries can be created and retrieved in this form.

For more information about display palettes, see `CGDirectPaletteRef` (page 1546).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectPalette.h

CGDirectDisplayID

Represents a unique identifier for an attached display.

```
typedef uint32_t CGDirectDisplayID;
```

Discussion
In Quartz, the term *display* refers to a graphics hardware system consisting of a framebuffer, a color correction (gamma) table or color palette, and possibly an attached monitor. If no monitor is attached, a display is characterized as offline.

When a monitor is attached, Quartz assigns a unique display identifier (ID). A display ID can persist across processes and system reboot, and typically remains constant as long as certain display parameters do not change.

When assigning a display ID, Quartz considers the following parameters:

- vendor
- model
- serial number
- position in the I/O Kit registry

For information about how to obtain a display ID, see "Finding Displays" (page 1470).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGDirectPaletteRef

Defines a reference to a Quartz 8-bit display palette.

```
typedef struct _CGDirectPaletteRef * CGDirectPaletteRef;
```

Discussion
A display palette is a bounded set of color values available for display. Some display operating modes have a maximum color depth of 8 bits (256 colors). The CGDirectPalette API is designed for application and game developers that want to create and use display palettes for these older displays.

Quartz uses reference counting to manage display palettes. See "Working With Color Palettes" (page 1474) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGDisplayBlendFraction

Represents the percentage of blend color used in a fade operation.

```
typedef float CGDisplayBlendFraction;
```

Discussion
The blend fraction ranges from 0 (no color) to 1 (full intensity). If you specify 0, the blend color is not applied. If you specify 1, the user sees only the blend color on the screen.

In a fade operation, Quartz blends a color specified by the application with the current contents of the frame buffer. The blend color can be applied both at the beginning and the end of a fade operation.

Color blending during a fade operation is analogous to alpha blending in Quartz 2D, and the visual appearance is similar. However, the implementation is quite different. In a fade operation, the blend color is applied at the very end of the graphics pipeline, as the frame buffer is transferred to video output.

For example, the Universal Access preference panel in Mac OS X allows you to select a flashing screen effect (sometimes called a visual bell) to accompany the system alert sound. When you select this option, the system uses a Quartz fade operation to produce the flash. The blend color is applied using a blend fraction of 0.5 or 50%.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGDisplayConfigRef

Defines a reference to a display configuration transaction.

```
typedef struct _CGDisplayConfigRef * CGDisplayConfigRef;
```

Discussion
This data type makes it possible to

- create a new display configuration transaction using the function `CGBeginDisplayConfiguration` (page 1477)
- record a set of configuration changes, each bound to one or more displays
- apply the changes in a single transaction using the function `CGCompleteDisplayConfiguration` (page 1479), or discard the changes using the function `CGCancelDisplayConfiguration` (page 1478)

There are no restrictions on the order in which you accumulate configuration changes in a transaction.

Configuration changes sometimes conflict with each other. For example, a new origin might be rendered invalid by a subsequent configuration change.

If possible, Quartz uses a “best fit” strategy to resolve conflicts between configuration changes. For example, when you change the resolution of a single display in a two-display system, Quartz automatically re-tiles the displays to prevent separation or overlap of the adjoining edges.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CGDisplayConfiguration.h

CGDisplayCoord

Represents a coordinate position in global display space.

```
typedef int32_t CGDisplayCoord;
```

Discussion
Quartz uses `CGDisplayCoord` to represent the x- and y-coordinates of points in the per-display coordinate system. The origin is defined as the upper-left corner of the screen.

This data type is also used in functions that need to find the address of a specific pixel and monitor. For example, the function `CGDisplayAddressForPosition` (page 1485) finds the address in frame buffer memory that corresponds to a given position or point.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGDisplayCount

Represents the number of displays in various lists.

```
typedef uint32_t CGDisplayCount;
```

Discussion
Quartz uses `CGDisplayCount` to represent a count of either the current or the maximum number of displays in a display list. For example, see the function `CGGetActiveDisplayList` (page 1515).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGDisplayErr

Defines a uniform type for result codes returned by functions in Quartz Display Services.

```
typedef CGError CGDisplayErr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGDirectDisplay.h

CGDisplayFadeInterval

Represents the duration in seconds of a fade operation or a fade hardware reservation.

```
typedef float CGDisplayFadeInterval;
```

Discussion
Quartz uses this data type to specify the duration of both fade-out and fade-in operations. Values may range from zero to `kCGMaxDisplayReservationInterval` seconds. A zero value means fade immediately—see `CGDisplayFade` (page 1495).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGDisplayFadeReservationToken

Defines a token issued by Quartz when reserving one or more displays for a fade operation during a specified interval.

```
typedef uint32_t CGDisplayFadeReservationToken;
```

Discussion
Quartz lets you reserve the display hardware to perform a fade operation. Fade reservations are valid for up to 15 seconds. Only one token is needed for both fade-out and fade-in.

You should release a fade reservation immediately when you no longer need it. If the reservation expires, releasing it is safe but not necessary.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGDisplayReservationInterval

Represents the time interval for a fade reservation.

```
typedef float CGDisplayReservationInterval;
```

Discussion
A fade reservation interval is a period of time during which a specific display is reserved for a fade operation. Fade reservation intervals range from 1 to `kCGMaxDisplayReservationInterval` seconds.

For more information about fade reservations, see the function `CGAcquireDisplayFadeReservation` (page 1476). Fade reservation tokens are discussed in `CGDisplayFadeReservationToken` (page 1549).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CGDisplayFade.h`

CGError

Defines a uniform type for result codes returned by functions in Quartz Services.

```
typedef int32_t CGError;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGError.h`

CGGammaValue

Represents information used to map a color generated in software to a color supported by the display hardware.

```
typedef float CGGammaValue;
```

Discussion
In Mac OS X, the Display panel in System Preferences is used to set the default gamma for a display. Quartz also allows an application to provide its own custom gamma information, using functions such as `CGSetDisplayTransferByTable` (page 1535) and `CGSetDisplayTransferByFormula` (page 1533).

These functions take `CGGammaValue` arguments that specify

- a set of gamma table entries ranging from 0 to 1
- the positive real coefficients in a gamma equation

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGMouseDelta

Represents a change in mouse position, in mouse units.

```
typedef int32_t CGMouseDelta;
```

Discussion
A mouse unit is a hardware-specific unit of measure, and generally has higher resolution than pixel units.

Note that the function `CGGetLastMouseDelta` (page 1520) is no longer recommended—instead, you should use mouse tracking functions in the Carbon Event Manager.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGOpenGLDisplayMask

Defines a bitmask used in OpenGL to specify a set of attached displays.

```
typedef uint32_t CGOpenGLDisplayMask;
```

Discussion
In Mac OS X, OpenGL can provide information about the capabilities of the hardware renderers driving a specified set of displays. A 32-bit mask is used to specify the displays—each bit in the mask represents a single display.

To learn how to find the mask bit that corresponds to a given display, see the function `CGDisplayIDToOpenGLDisplayMask` (page 1498).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGPaletteBlendFraction

Represents the intensity of a solid color used to tint a display palette.

```
typedef float CGPaletteBlendFraction;
```

Discussion
A palette blend-fraction value can range from 0 (no color) to 1 (full intensity). At full intensity, the palette is completely washed out by the color.

For more information, see the function `CGPaletteCreateFromPaletteBlendedWithColor` (page 1524).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectPalette.h`

CGRectCount

Represents the size of an array of Quartz rectangles.

```
typedef uint32_t CGRectCount;
```

Discussion
For example, see the function `CGWaitForScreenRefreshRects` (page 1537).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGRemoteOperation.h`

CGRefreshRate

Represents a display's refresh rate in frames per second.

```
typedef double CGRefreshRate;
```

Discussion
When requesting a new display mode, you can specify a desired refresh rate as a hint to Quartz. For example, see the function `CGDisplayBestModeForParametersAndRefreshRate` (page 1488).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGScreenUpdateMoveDelta

Represents the distance a region on the screen moves in pixel units.

```
struct _CGScreenUpdateMoveDelta {
      int32_t dX, dY;
};
typedef struct _CGScreenUpdateMoveDelta CGScreenUpdateMoveDelta;
```

Discussion
Move operation notifications are restricted to changes that move a region by an integer number of pixels. The fields `dX` and `dY` describe the direction of movement:

- Positive values of `dX` indicate movement to the right.
- Negative values of `dX` indicate movement to the left.
- Positive values of `dY` indicate movement downward.
- Negative values of `dY` indicate movement upward.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CGRemoteOperation.h`

CGTableCount

Defines a uniform type to represent the number of entries in a table.

```
typedef uint32_t CGTableCount;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGDirectDisplay.h`

CGWindowLevel

Represents a level assigned to a window by an application framework.

```
typedef int32_t CGWindowLevel;
```

Discussion
In Mac OS X, application frameworks support the concept of multiple window levels (or layers). Window levels are assigned and managed by each individual framework.

Note that in an Aqua-compliant application, each document window exists in its own layer. As a result, windows created by different applications can be interleaved.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGWindowLevel.h`

Constants

Display Capture Options

Specify configuration parameters when capturing displays.

```
enum {
    kCGCaptureNoOptions = 0,
    kCGCaptureNoFill = (1 << 0)
};
typedef uint32_t CGCaptureOptions;
```

Constants

`kCGCaptureNoOptions`
Specifies that the system should use the default fill behavior, which is fill with black.
Available in Mac OS X v10.3 and later.
Declared in `CGDirectDisplay.h`.

`kCGCaptureNoFill`
Disables fill with black.
Available in Mac OS X v10.3 and later.
Declared in `CGDirectDisplay.h`.

Discussion
For information about how these constants are used, see the functions `CGDisplayCaptureWithOptions` (page 1493) and `CGCaptureAllDisplaysWithOptions` (page 1479).

Display Configuration Change Flags

Specify the configuration parameters passed to a display reconfiguration callback function.

```
enum {
    kCGDisplayBeginConfigurationFlag = (1 << 0),
    kCGDisplayMovedFlag              = (1 << 1),
    kCGDisplaySetMainFlag            = (1 << 2),
    kCGDisplaySetModeFlag            = (1 << 3),
    kCGDisplayAddFlag                = (1 << 4),
    kCGDisplayRemoveFlag             = (1 << 5),
    kCGDisplayEnabledFlag            = (1 << 8),
    kCGDisplayDisabledFlag           = (1 << 9),
    kCGDisplayMirrorFlag             = (1 << 10),
    kCGDisplayUnMirrorFlag           = (1 << 11),
    kCGDisplayDesktopShapeChangedFlag = (1 << 12)
};
typedef u_int32_t CGDisplayChangeSummaryFlags;
```

Constants

`kCGDisplayBeginConfigurationFlag`
The display configuration is about to change.
Available in Mac OS X v10.3 and later.
Declared in `CGDisplayConfiguration.h`.

`kCGDisplayMovedFlag`
The location of the upper-left corner of the display in global display space has changed.
Available in Mac OS X v10.3 and later.
Declared in `CGDisplayConfiguration.h`.

`kCGDisplaySetMainFlag`
The display is now the main display.
Available in Mac OS X v10.3 and later.
Declared in `CGDisplayConfiguration.h`.

`kCGDisplaySetModeFlag`
The display mode has changed.
Available in Mac OS X v10.3 and later.
Declared in `CGDisplayConfiguration.h`.

`kCGDisplayAddFlag`
The display has been added to the active display list.
Available in Mac OS X v10.3 and later.
Declared in `CGDisplayConfiguration.h`.

`kCGDisplayRemoveFlag`
The display has been removed from the active display list.
Available in Mac OS X v10.3 and later.
Declared in `CGDisplayConfiguration.h`.

`kCGDisplayEnabledFlag`
The display has been enabled.
Available in Mac OS X v10.3 and later.
Declared in `CGDisplayConfiguration.h`.

`kCGDisplayDisabledFlag`
The display has been disabled.

Available in Mac OS X v10.3 and later.

Declared in `CGDisplayConfiguration.h`.

`kCGDisplayMirrorFlag`
The display is now mirroring another display.

Available in Mac OS X v10.3 and later.

Declared in `CGDisplayConfiguration.h`.

`kCGDisplayUnMirrorFlag`
The display is no longer mirroring another display.

Available in Mac OS X v10.3 and later.

Declared in `CGDisplayConfiguration.h`.

`kCGDisplayDesktopShapeChangedFlag`
The shape of the desktop (the union of display areas) has changed.

Available in Mac OS X v10.5 and later.

Declared in `CGDisplayConfiguration.h`.

Discussion
For information about how these constants are used, see the callback `CGDisplayReconfigurationCallBack` (page 1540).

Display Configuration Scopes

Specify the scope of the changes in a display configuration transaction.

```
enum {
    kCGConfigureForAppOnly = 0,
    kCGConfigureForSession = 1,
    kCGConfigurePermanently = 2
};
```

Constants
`kCGConfigureForAppOnly`
Specifies that changes persist for the lifetime of the current application. After the application terminates, the display configuration settings revert to the current login session.

Available in Mac OS X v10.2 and later.

Declared in `CGDisplayConfiguration.h`.

`kCGConfigureForSession`
Specifies that changes persist for the lifetime of the current login session. After the current session terminates, the displays revert to the last saved permanent configuration.

Available in Mac OS X v10.2 and later.

Declared in `CGDisplayConfiguration.h`.

`kCGConfigurePermanently`
Specifies that changes persist in future login sessions by the same user. If the requested changes cannot be supported by the Aqua UI (resolution and pixel depth constraints apply), the settings for the current login session are used instead, and any changes have session scope.

Available in Mac OS X v10.2 and later.

Declared in `CGDisplayConfiguration.h`.

Discussion
For information about how these constants are used, see the function `CGCompleteDisplayConfiguration` (page 1479).

Display Fade Blend Fractions

Specify the lower and upper bounds for blend color fractions during a display fade operation.

```
#define kCGDisplayBlendNormal (0.0)
#define kCGDisplayBlendSolidColor (1.0)
```

Constants
`kCGDisplayBlendNormal`
Specifies that the blend color is not applied at the start or end of a fade operation.

Available in Mac OS X v10.2 and later.

Declared in `CGDisplayFade.h`.

`kCGDisplayBlendSolidColor`
Specifies that the user sees only the blend color at the start or end of a fade operation.

Available in Mac OS X v10.2 and later.

Declared in `CGDisplayFade.h`.

Discussion
For general information about blend fractions, see the data type `CGDisplayBlendFraction` (page 1547). For information about how these constants are used, see the function `CGDisplayFade` (page 1495).

Display Fade Constants

Specifies values relating to fade operations.

```
#define kCGMaxDisplayReservationInterval (15.0)
#define kCGDisplayFadeReservationInvalidToken (0)
```

Constants
`kCGMaxDisplayReservationInterval`
Specifies the maximum number of seconds for fade hardware reservations and display fade operations. For general information about fade intervals, see the data type `CGDisplayFadeInterval` (page 1549).

Available in Mac OS X v10.2 and later.

Declared in `CGDisplayFade.h`.

`kCGDisplayFadeReservationInvalidToken`
Specifies an invalid fade reservation token. For general information about fade reservation tokens, see the data type `CGDisplayFadeReservationToken` (page 1549).

Available in Mac OS X v10.2 and later.

Declared in `CGDisplayFade.h`.

Display ID Defaults

Default values for a display ID.

```
#define kCGDirectMainDisplay (CGMainDisplayID())
#define kCGNullDirectDisplay ((CGDirectDisplayID)0)
```

Constants

`kCGDirectMainDisplay`
Specifies the current main display ID.

Available in Mac OS X v10.0 and later.

Declared in `CGDirectDisplay.h`.

`kCGNullDirectDisplay`
Specifies a value that will never correspond to actual hardware.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

Display Mode Standard Properties

Specify keys for the standard properties in a display mode dictionary.

```
#define kCGDisplayWidth CFSTR("Width")
#define kCGDisplayHeight CFSTR("Height")
#define kCGDisplayMode CFSTR("Mode")
#define kCGDisplayBitsPerPixel CFSTR("BitsPerPixel")
#define kCGDisplayBitsPerSample CFSTR("BitsPerSample")
#define kCGDisplaySamplesPerPixel CFSTR("SamplesPerPixel")
#define kCGDisplayRefreshRate CFSTR("RefreshRate")
#define kCGDisplayModeUsableForDesktopGUICFSTR("UsableForDesktopGUI")
#define kCGDisplayIOFlags CFSTR("IOFlags")
#define kCGDisplayBytesPerRow CFSTR("kCGDisplayBytesPerRow")
```

Constants

`kCGDisplayWidth`
Specifies a CFNumber integer value that represents the width of the display in pixels.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayHeight`
Specifies a CFNumber integer value that represents the height of the display in pixels.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayMode`
Specifies a CFNumber integer value that represents the I/O Kit display mode number.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayBitsPerPixel`
Specifies a CFNumber integer value that represents the number of bits in a pixel.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayBitsPerSample`
Specifies a CFNumber integer value that represents the number of bits in an individual sample (for example, a color value in an RGB pixel).

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplaySamplesPerPixel`
Specifies a CFNumber integer value that represents the number of samples in a pixel.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayRefreshRate`
Specifies a CFNumber double-precision floating point value that represents the refresh rate of a CRT display. Some displays may not use conventional video vertical and horizontal sweep in painting the screen; these displays report a refresh rate of 0.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayModeUsableForDesktopGUI`
Specifies a CFBoolean value that indicates whether the display is suitable for use with the Mac OS X graphical user interface. The criteria include factors such as sufficient width and height and adequate pixel depth.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayIOFlags`
Specifies a CFNumber integer value that contains the I/O Kit display mode flags. For more information, see the header file `IOKit/IOGraphicsTypes.h`.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayBytesPerRow`
Specifies a CFNumber integer value that represents the number of bytes in a row on the display.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

Discussion
To learn how to use these keys to access the values in a display mode dictionary, see *CFDictionary Reference*.

Display Mode Optional Properties

Specify keys for optional properties in a display mode dictionary.

```
#define kCGDisplayModeIsSafeForHardware CFSTR ("kCGDisplayModeIsSafeForHardware")
#define kCGDisplayModeIsInterlaced CFSTR("kCGDisplayModeIsInterlaced")
#define kCGDisplayModeIsStretched CFSTR("kCGDisplayModeIsStretched")
#define kCGDisplayModeIsTelevisionOutput CFSTR ("kCGDisplayModeIsTelevisionOutput")
```

Constants

`kCGDisplayModeIsSafeForHardware`

Specifies a CFBoolean value indicating that the display mode doesn't need a confirmation dialog to be set.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayModeIsInterlaced`

Specifies a CFBoolean value indicating that the I/O Kit interlace mode flag is set.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayModeIsStretched`

Specifies a CFBoolean value indicating that the I/O Kit stretched mode flag is set.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

`kCGDisplayModeIsTelevisionOutput`

Specifies a CFBoolean value indicating that the I/O Kit television output mode flag is set.

Available in Mac OS X v10.2 and later.

Declared in `CGDirectDisplay.h`.

Discussion

A given key is present in a display mode dictionary only if the property applies, and is always associated with a value of `kCFBooleanTrue`. Keys not relevant to a particular display mode will not appear in the mode dictionary.

Reserved Window Levels

Specifies window level constants.

```
#define kCGNumReservedWindowLevels      (16)
```

Constants

`kCGNumReservedWindowLevels`

The number of window levels reserved by Apple for internal use. Application frameworks such as Carbon and Cocoa use this constant during compilation

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

Screen Update Operations

Specify types of screen update operations.

```
enum _CGScreenUpdateOperation {
    kCGScreenUpdateOperationRefresh = 0,
    kCGScreenUpdateOperationMove = (1 << 0),
    kCGScreenUpdateOperationReducedDirtyRectangleCount = (1 <<  31)
};
typedef uint32_t CGScreenUpdateOperation;
```

Constants

`kCGScreenUpdateOperationRefresh`
Specifies a screen refresh operation.
Available in Mac OS X v10.3 and later.
Declared in `CGRemoteOperation.h`.

`kCGScreenUpdateOperationMove`
Specifies a screen move operation.
Available in Mac OS X v10.3 and later.
Declared in `CGRemoteOperation.h`.

`kCGScreenUpdateOperationReducedDirtyRectangleCount`
When presented as part of the requested operations to the function `CGWaitForScreenUpdateRects` (page 1538), specifies that the function should try to minimize the number of rectangles returned to represent the changed areas of the display. The function may combine adjacent rectangles within a larger bounding rectangle, which may include unmodified areas of the display.
Available in Mac OS X v10.4 and later.
Declared in `CGRemoteOperation.h`.

Discussion

For information about how these constants are used, see the function `CGWaitForScreenUpdateRects` (page 1538).

Window Level Keys

Keys that represent the standard window levels in Mac OS X. Quartz includes these keys to support application frameworks such as Carbon and Cocoa. Applications do not need to use them directly.

```
enum _CGCommonWindowLevelKey {
 kCGBaseWindowLevelKey = 0,
 kCGMinimumWindowLevelKey,
 kCGDesktopWindowLevelKey,
 kCGBackstopMenuLevelKey,
 kCGNormalWindowLevelKey,
 kCGFloatingWindowLevelKey,
 kCGTornOffMenuWindowLevelKey,
 kCGDockWindowLevelKey,
 kCGMainMenuWindowLevelKey,
 kCGStatusWindowLevelKey,
 kCGModalPanelWindowLevelKey,
 kCGPopUpMenuWindowLevelKey,
 kCGDraggingWindowLevelKey,
 kCGScreenSaverWindowLevelKey,
 kCGMaximumWindowLevelKey,
 kCGOverlayWindowLevelKey,
 kCGHelpWindowLevelKey,
 kCGUtilityWindowLevelKey,
 kCGDesktopIconWindowLevelKey,
 kCGCursorWindowLevelKey,
 kCGNumberOfWindowLevelKeys
};
typedef int32_t CGWindowLevelKey;
```

Constants

`kCGBaseWindowLevelKey`
The base key used to define window levels. Do not use.
Available in Mac OS X v10.0 and later.
Declared in `CGWindowLevel.h`.

`kCGMinimumWindowLevelKey`
The lowest available window level.
Available in Mac OS X v10.0 and later.
Declared in `CGWindowLevel.h`.

`kCGDesktopWindowLevelKey`
The level for the desktop.
Available in Mac OS X v10.0 and later.
Declared in `CGWindowLevel.h`.

`kCGBackstopMenuLevelKey`
The level of the backstop menu.
Available in Mac OS X v10.0 and later.
Declared in `CGWindowLevel.h`.

`kCGNormalWindowLevelKey`
The level for normal windows.
Available in Mac OS X v10.0 and later.
Declared in `CGWindowLevel.h`.

`kCGFloatingWindowLevelKey`
The level for floating windows.
Available in Mac OS X v10.0 and later.
Declared in `CGWindowLevel.h`.

`kCGTornOffMenuWindowLevelKey`
The level for torn off menus.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGDockWindowLevelKey`
The level for the dock.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGMainMenuWindowLevelKey`
The level for the menus displayed in the menu bar.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGStatusWindowLevelKey`
The level for status windows.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGModalPanelWindowLevelKey`
The level for modal panels.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGPopUpMenuWindowLevelKey`
The level for pop-up menus.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGDraggingWindowLevelKey`
The level for a window being dragged.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGScreenSaverWindowLevelKey`
The level for screen savers.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGMaximumWindowLevelKey`
The highest allowed window level.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

`kCGOverlayWindowLevelKey`
The level for overlay windows.

Available in Mac OS X v10.1 and later.

Declared in `CGWindowLevel.h`.

`kCGHelpWindowLevelKey`
The level for help windows.

Available in Mac OS X v10.1 and later.

Declared in `CGWindowLevel.h`.

`kCGUtilityWindowLevelKey`
The level for utility windows.

Available in Mac OS X v10.1 and later.

Declared in `CGWindowLevel.h`.

`kCGDesktopIconWindowLevelKey`
The level for desktop icons.

Available in Mac OS X v10.1 and later.

Declared in `CGWindowLevel.h`.

`kCGCursorWindowLevelKey`
The level for the cursor.

Available in Mac OS X v10.2 and later.

Declared in `CGWindowLevel.h`.

`kCGNumberOfWindowLevelKeys`
The total number of window levels.

Available in Mac OS X v10.0 and later.

Declared in `CGWindowLevel.h`.

Window Server Session Properties

Specify keys for the standard properties in a window server session dictionary.

```
#define kCGSessionUserIDKey CFSTR("kCGSSessionUserIDKey")
#define kCGSessionUserNameKey CFSTR("kCGSSessionUserNameKey")
#define kCGSessionConsoleSetKey CFSTR("kCGSSessionConsoleSetKey")
#define kCGSessionOnConsoleKey CFSTR("kCGSSessionOnConsoleKey")
#define kCGSessionLoginDoneKey CFSTR("kCGSessionLoginDoneKey")
```

Constants

`kCGSessionUserIDKey`
Specifies a CFNumber 32-bit unsigned integer value that encodes a user ID for the session's current user.

Available in Mac OS X v10.3 and later.

Declared in `CGSession.h`.

`kCGSessionUserNameKey`
Specifies a CFString value that encodes the session's short user name as set by the login operation.

Available in Mac OS X v10.3 and later.

Declared in `CGSession.h`.

`kCGSessionConsoleSetKey`
: Specifies a CFNumber 32-bit unsigned integer value that represents a set of hardware composing a console.

Available in Mac OS X v10.3 and later.

Declared in `CGSession.h`.

`kCGSessionOnConsoleKey`
: Specifies a CFBoolean value indicating whether the session is on a console.

Available in Mac OS X v10.3 and later.

Declared in `CGSession.h`.

`kCGSessionLoginDoneKey`
: Specifies a CFBoolean value indicating whether the login operation has been done.

Available in Mac OS X v10.3 and later.

Declared in `CGSession.h`.

Discussion
To learn how to use these keys to access the values in a session dictionary, see *CFDictionary Reference*.

Result Codes

This table lists the result codes returned by functions in Quartz Display Services.

Result Code	Value	Description
`kCGErrorSuccess`	0	The requested operation was completed successfully. Available in Mac OS X v10.0 and later.
`kCGErrorFailure`	1000	A general failure occurred. Available in Mac OS X v10.0 and later.
`kCGErrorIllegalArgument`	1001	One or more of the parameters passed to a function are invalid. Check for `NULL` pointers. Available in Mac OS X v10.0 and later.
`kCGErrorInvalidConnection`	1002	The parameter representing a connection to the window server is invalid. Available in Mac OS X v10.0 and later.
`kCGErrorInvalidContext`	1003	The `CPSProcessSerNum` or context identifier parameter is not valid. Available in Mac OS X v10.0 and later.
`kCGErrorCannotComplete`	1004	The requested operation is inappropriate for the parameters passed in, or the current system state. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kCGErrorNameTooLong	1005	A parameter, typically a C string, is too long to be used without truncation. Available in Mac OS X v10.0 and later.
kCGErrorNotImplemented	1006	Return value from obsolete function stubs present for binary compatibility, but not normally called. Available in Mac OS X v10.0 and later.
kCGErrorRangeCheck	1007	A parameter passed in has a value that is inappropriate, or which does not map to a useful operation or value. Available in Mac OS X v10.0 and later.
kCGErrorTypeCheck	1008	A data type or token was encountered that did not match the expected type or token. Available in Mac OS X v10.0 and later.
kCGErrorNoCurrentPoint	1009	An operation relative to a known point or coordinate could not be done, as there is no known point. Available in Mac OS X v10.0 and later.
kCGErrorInvalidOperation	1010	The requested operation is not valid for the parameters passed in, or the current system state. Available in Mac OS X v10.0 and later.
kCGErrorNoneAvailable	1011	The requested operation could not be completed as the indicated resources were not found. Available in Mac OS X v10.0 and later.

Quartz Event Services Reference

Framework: ApplicationServices/ApplicationServices.h

Declared in CGEvent.h
CGEventSource.h
CGEventTypes.h
CGRemoteOperation.h

Overview

This document describes the C API for event taps, which are filters used to observe and alter the stream of low-level user input events in Mac OS X. Event taps make it possible to monitor and filter input events from several points within the system, prior to their delivery to a foreground application. Event taps complement and extend the capabilities of the Carbon event monitor mechanism, which allows an application to observe input events delivered to other processes (see the function `GetEventMonitorTarget`).

Event taps are designed to serve as a Section 508 enabling technology. For example, consider a software system to assist a person with language impairments, designed to perform keyboard filtering with spoken review. Such a system could use an event tap to monitor all keystrokes, perform dictionary checks and matches, and recite the assembled word back to the user on detection of a word break in the input stream. If acceptable to the user, as indicated by an additional input keystroke or other gesture, the events would be posted into the system for delivery to the foreground application.

Introduced in Mac OS X version 10.4, event taps provide functionality similar to the Win32 functions `SetWinEventHook` when used to establish an out-of-context event hook, and `SendInput`. Quartz Event Services also includes an older set of event-related functions declared in the file `CGRemoteOperation.h`. These functions are still supported, but they are not recommended for new development.

Functions by Task

Working With Quartz Events

`CGEventGetTypeID` (page 1579)
Returns the type identifier for the opaque type `CGEventRef`.

`CGEventCreate` (page 1571)
Returns a new Quartz event.

`CGEventCreateData` (page 1572)
Returns a flattened data representation of a Quartz event.

`CGEventCreateFromData` (page 1573)
Returns a Quartz event created from a flattened data representation of the event.

`CGEventCreateMouseEvent` (page 1574)
Returns a new Quartz mouse event.

`CGEventCreateKeyboardEvent` (page 1573)
Returns a new Quartz keyboard event.

`CGEventCreateScrollWheelEvent` (page 1575)
Returns a new Quartz scrolling event.

`CGEventCreateCopy` (page 1572)
Returns a copy of an existing Quartz event.

`CGEventCreateSourceFromEvent` (page 1576)
Returns a Quartz event source created from an existing Quartz event.

`CGEventSetSource` (page 1585)
Sets the event source of a Quartz event.

`CGEventGetType` (page 1579)
Returns the event type of a Quartz event (left mouse down, for example).

`CGEventSetType` (page 1586)
Sets the event type of a Quartz event (left mouse down, for example).

`CGEventGetTimestamp` (page 1578)
Returns the timestamp of a Quartz event.

`CGEventSetTimestamp` (page 1585)
Sets the timestamp of a Quartz event.

`CGEventGetLocation` (page 1578)
Returns the location of a Quartz mouse event.

`CGEventGetUnflippedLocation` (page 1579)
Returns the location of a Quartz mouse event.

`CGEventGetFlags` (page 1577)
Returns the event flags of a Quartz event.

`CGEventSetFlags` (page 1583)
Sets the event flags of a Quartz event.

`CGEventKeyboardGetUnicodeString` (page 1580)
Returns the Unicode string associated with a Quartz keyboard event.

`CGEventKeyboardSetUnicodeString` (page 1581)
Sets the Unicode string associated with a Quartz keyboard event.

`CGEventGetIntegerValueField` (page 1577)
Returns the integer value of a field in a Quartz event.

`CGEventSetIntegerValueField` (page 1584)
Sets the integer value of a field in a Quartz event.

`CGEventGetDoubleValueField` (page 1576)
Returns the floating-point value of a field in a Quartz event.

`CGEventSetDoubleValueField` (page 1583)
Sets the floating-point value of a field in a Quartz event.

`CGEventSetLocation` (page 1584) Deprecated in Mac OS X v10.5
Sets the location of a Quartz mouse event.

Working With Quartz Event Taps

`CGEventTapCreate` (page 1595)
Creates an event tap.

`CGEventTapCreateForPSN` (page 1597)
Creates an event tap for a specified process.

`CGEventTapEnable` (page 1598)
Enables or disables an event tap.

`CGEventTapIsEnabled` (page 1598)
Returns a Boolean value indicating whether an event tap is enabled.

`CGEventTapPostEvent` (page 1599)
Posts a Quartz event from an event tap into the event stream.

`CGEventPost` (page 1582)
Posts a Quartz event into the event stream at a specified location.

`CGEventPostToPSN` (page 1582)
Posts a Quartz event into the event stream for a specific application.

`CGGetEventTapList` (page 1599)
Gets a list of currently installed event taps.

`CGEventMaskBit` (page 1581)
Generates an event mask for a single type of event.

Working With Quartz Event Sources

`CGEventSourceGetTypeID` (page 1591)
Returns the type identifier for the opaque type `CGEventSourceRef`.

`CGEventSourceCreate` (page 1587)
Returns a Quartz event source created with a specified source state.

`CGEventSourceGetKeyboardType` (page 1588)
Returns the keyboard type to be used with a Quartz event source.

`CGEventSourceSetKeyboardType` (page 1593)
Sets the keyboard type to be used with a Quartz event source.

`CGEventSourceGetSourceStateID` (page 1590)
Returns the source state associated with a Quartz event source.

`CGEventSourceButtonState` (page 1586)
Returns a Boolean value indicating the current button state of a Quartz event source.

`CGEventSourceKeyState` (page 1592)
Returns a Boolean value indicating the current keyboard state of a Quartz event source.

`CGEventSourceFlagsState` (page 1588)
Returns the current flags of a Quartz event source.

`CGEventSourceSecondsSinceLastEventType` (page 1592)
Returns the elapsed time since the last event for a Quartz event source.

`CGEventSourceCounterForEventType` (page 1587)
Returns a count of events of a given type seen since the window server started.

`CGEventSourceGetUserData` (page 1591)
Returns the 64-bit user-specified data for a Quartz event source.

`CGEventSourceSetUserData` (page 1595)
Sets the 64-bit user-specified data for a Quartz event source.

`CGEventSourceGetLocalEventsFilterDuringSuppressionState` (page 1589)
Returns the mask that indicates which classes of local hardware events are enabled during event suppression.

`CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 1593)
Sets the mask that indicates which classes of local hardware events are enabled during event suppression.

`CGEventSourceSetLocalEventsSuppressionInterval` (page 1594)
Sets the interval that local hardware events may be suppressed following the posting of a Quartz event.

`CGEventSourceGetPixelsPerLine` (page 1590)
Gets the scale of pixels per line in a scrolling event source.

`CGEventSourceSetPixelsPerLine` (page 1594)
Sets the scale of pixels per line in a scrolling event source.

`CGEventSourceGetLocalEventsSuppressionInterval` (page 1589) Deprecated in Mac OS X v10.4
Returns the interval that local hardware events may be suppressed following the posting of a Quartz event.

Obsolete Functions

`CGPostKeyboardEvent` (page 1600)
Synthesizes a low-level keyboard event on the local machine.

`CGPostMouseEvent` (page 1601)
Synthesizes a low-level mouse-button event on the local machine.

`CGPostScrollWheelEvent` (page 1602)
Synthesizes a low-level scrolling event on the local machine.

`CGEnableEventStateCombining` (page 1571)
Enables or disables the merging of actual key and mouse state with the application-specified state in a synthetic event.

`CGInhibitLocalEvents` (page 1600)
Turns off local hardware events in the current session.

`CGSetLocalEventsFilterDuringSuppressionState` (page 1603)
Filters local hardware events from the keyboard and mouse during the short interval after a synthetic event is posted.

`CGSetLocalEventsSuppressionInterval` (page 1603)
Sets the time interval in seconds that local hardware events are suppressed after posting a synthetic event.

`CGEventGetSource` (page 1578) Deprecated in Mac OS X v10.4
Returns a Quartz event source created from an existing Quartz event. (Deprecated. Use `CGEventCreateSourceFromEvent` (page 1576) instead.)

Functions

CGEnableEventStateCombining

Enables or disables the merging of actual key and mouse state with the application-specified state in a synthetic event.

```
CGError CGEnableEventStateCombining (
   boolean_t doCombineState
);
```

Parameters

doCombineState

Pass `true` to specify that the actual key and mouse state are merged with the application-specified state in a synthetic event; otherwise, pass `false`.

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion

By default, the flags that indicate modifier key state (Command, Option, Shift, Control, and so on) from the system's keyboard and from other event sources are ORed together as an event is posted into the system, and current key and mouse button state is considered in generating new events. This function allows your application to enable or disable the merging of event state. When combining is turned off, the event state propagated in the events posted by your application reflect state built up only by your application. The state within your application's generated event will not be combined with the system's current state, so the system-wide state reflecting key and mouse button state will remain unchanged. When called with `doCombineState` equal to `false`, this function initializes local (per application) state tracking information to a state of all keys, modifiers, and mouse buttons up. When called with `doCombineState` equal to `true`, the current global state of keys, modifiers, and mouse buttons are used in generating events.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is to use Quartz events and Quartz event sources. This allows you to control exactly which, if any, external event sources will contribute to the state used to create an event.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CGRemoteOperation.h`

CGEventCreate

Returns a new Quartz event.

```
CGEventRef CGEventCreate (
   CGEventSourceRef source
);
```

Parameters

source

The event source, or `NULL` to use a default source.

Return Value

A new event to be filled in, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateCopy

Returns a copy of an existing Quartz event.

```
CGEventRef CGEventCreateCopy (
   CGEventRef event
);
```

Parameters

event

The event being copied.

Return Value

A copy of the specified event. When you no longer need the copy, you should release it using the function `CFRelease`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateData

Returns a flattened data representation of a Quartz event.

```
CFDataRef CGEventCreateData (
   CFAllocatorRef allocator,
   CGEventRef event
);
```

Parameters

allocator

The allocator to use to allocate memory for the data object. To use the current default allocator, pass `NULL` or `kCFAllocatorDefault`.

event
The event to flatten.

Return Value
The flattened data representation of the event, or `NULL` if the `event` parameter is invalid. When you no longer need the data object, you should release it using the function `CFRelease`.

Discussion
You can use this function to flatten an event for network transport to another system.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventCreateFromData

Returns a Quartz event created from a flattened data representation of the event.

```
CGEventRef CGEventCreateFromData (
   CFAllocatorRef allocator,
   CFDataRef eventData
);
```

Parameters

allocator
The allocator to use to allocate memory for the event object. To use the current default allocator, pass `NULL` or `kCFAllocatorDefault`.

eventData
The flattened data representation of the event to reconstruct.

Return Value
An event built from the flattened data representation, or `NULL` if the `eventData` parameter is invalid.

Discussion
You can use this function to reconstruct a Quartz event received by network transport from another system.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventCreateKeyboardEvent

Returns a new Quartz keyboard event.

```
CGEventRef CGEventCreateKeyboardEvent (
   CGEventSourceRef source,
   CGKeyCode virtualKey,
   bool keyDown
);
```

Parameters

source

An event source taken from another event, or `NULL`.

virtualKey

The virtual key code for the event.

keyDown

Pass `true` to specify that the key position is down. To specify that the key position is up, pass `false`. This value is used to determine the type of the keyboard event—see "Event Types" (page 1623).

Return Value

A new keyboard event, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

Discussion

All keystrokes needed to generate a character must be entered, including modifier keys. For example, to produce a 'Z', the SHIFT key must be down, the 'z' key must go down, and then the SHIFT and 'z' key must be released:

```
CGEventRef event1, event2, event3, event4;
event1 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)56, true);
event2 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)6, true);
event3 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)6, false);
event4 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)56, false);
```

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateMouseEvent

Returns a new Quartz mouse event.

```
CGEventRef CGEventCreateMouseEvent (
   CGEventSourceRef source,
   CGEventType mouseType,
   CGPoint mouseCursorPosition,
   CGMouseButton mouseButton
);
```

Parameters

source

An event source taken from another event, or `NULL`.

mouseType

A mouse event type. Pass one of the constants listed in "Event Types" (page 1623).

mouseCursorPosition
The position of the mouse cursor in global coordinates.

mouseButton
The button that's changing state. Pass one of the constants listed in "Mouse Buttons" (page 1626). This parameter is ignored unless the *mouseType* parameter is `kCGEventOtherMouseDown`, `kCGEventOtherMouseDragged`, or `kCGEventOtherMouseUp`.

Return Value
A new mouse event, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventCreateScrollWheelEvent

Returns a new Quartz scrolling event.

```
CGEventRef CGEventCreateScrollWheelEvent (
    CGEventSourceRef source,
    CGScrollEventUnit units,
    CGWheelCount wheelCount,
    int32_t wheel1,

);
```

Parameters

source
An event source taken from another event, or `NULL`.

units
The unit of measurement for the scrolling event. Pass one of the constants listed in "Scrolling Event Units" (page 1627).

wheelCount
The number of scrolling devices on the mouse, up to a maximum of 3.

wheel1
A value that reflects the movement of the primary scrolling device on the mouse. Scrolling movement is generally represented by small signed integer values, typically in a range from -10 to +10. Large values may have unexpected results, depending on the application that processes the event.

...
Up to two values that reflect the movements of the other scrolling devices on the mouse, if any.

Return Value
A new scrolling event, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

Discussion
This function allows you to create a scrolling event and customize the event before posting it to the event system.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGEvent.h`

CGEventCreateSourceFromEvent

Returns a Quartz event source created from an existing Quartz event.

```
CGEventSourceRef CGEventCreateSourceFromEvent (
   CGEventRef event
);
```

Parameters

event

The event to access.

Return Value
An event source created from the specified event, or `NULL` if the event was generated with a private event source owned by another process. When you no longer need this event source, you should release it using the function `CFRelease`.

Discussion
Event filters may use the event source to generate events that are compatible with an event being filtered.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventGetDoubleValueField

Returns the floating-point value of a field in a Quartz event.

```
double CGEventGetDoubleValueField (
   CGEventRef event,
   CGEventField field
);
```

Parameters

event

The event to access.

field

A field in the specified event. Pass one of the constants listed in "Event Fields" (page 1610).

Return Value
A floating point representation of the current value of the specified field.

Discussion
In cases where the field value is represented within the event by a fixed point number or an integer, the result is scaled to the appropriate range as part of creating the floating point representation.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventGetFlags

Returns the event flags of a Quartz event.

```
CGEventFlags CGEventGetFlags (
   CGEventRef event
);
```

Parameters

event
The event to access.

Return Value
The current flags of the specified event. For more information, see "Event Flags" (page 1618).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventGetIntegerValueField

Returns the integer value of a field in a Quartz event.

```
int64_t CGEventGetIntegerValueField (
   CGEventRef event,
   CGEventField field
);
```

Parameters

event
The event to access.

field
A field in the specified event. Pass one of the constants listed in "Event Fields" (page 1610).

Return Value
A 64-bit integer representation of the current value of the specified field.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventGetLocation

Returns the location of a Quartz mouse event.

```
CGPoint CGEventGetLocation (
   CGEventRef event
);
```

Parameters

event

The mouse event to locate.

Return Value

The current location of the specified mouse event in global display coordinates.

Availability

Available in Mac OS X v10.4 and later.

Declared In

CGEvent.h

CGEventGetSource

Returns a Quartz event source created from an existing Quartz event. (Deprecated in Mac OS X v10.4. Use CGEventCreateSourceFromEvent (page 1576) instead.)

```
CGEventSourceRef CGEventGetSource (
    CGEventRef event
);
```

Availability

Available in Mac OS X v10.4 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

CGEvent.h

CGEventGetTimestamp

Returns the timestamp of a Quartz event.

```
CGEventTimestamp CGEventGetTimestamp (
   CGEventRef event
);
```

Parameters

event

The event to access.

Return Value

The current timestamp of the specified event.

Availability

Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventGetType

Returns the event type of a Quartz event (left mouse down, for example).

```
CGEventType CGEventGetType (
   CGEventRef event
);
```

Parameters

event
The event to access.

Return Value
The current event type of the specified event. The return value is one of the constants listed in "Event Types" (page 1623).

Availability
Available in Mac OS X v10.4 and later.

See Also
CGEventSetType (page 1586)

Declared In
CGEvent.h

CGEventGetTypeID

Returns the type identifier for the opaque type CGEventRef.

```
CFTypeID CGEventGetTypeID (
   void
);
```

Return Value
The Core Foundation type identifier for the opaque type CGEventRef (page 1606).

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventGetUnflippedLocation

Returns the location of a Quartz mouse event.

```
CGPoint CGEventGetUnflippedLocation (
    CGEventRef event
);
```

Parameters

event

The mouse event whose location you wish to obtain.

Return Value

The current location of the specified mouse event relative to the lower-left corner of the main display.

Discussion

This function returns the location of the mouse cursor associated with the event. The coordinate system used is relative to the lower-left corner of the main display, and is compatible with the global coordinate system used by the Application Kit.

Note that the y-coordinate of the returned location is off by one from an idealized coordinate system originating at the lower-left corner of the main display. Effectively, the function is defined as follows:

```
CGPoint p = CGEventGetLocation(event);
p.y = main_display_height - p.y;
/* not p.y = (main_display_height - 1) - p.y */
return p;
```

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CGEvent.h`

CGEventKeyboardGetUnicodeString

Returns the Unicode string associated with a Quartz keyboard event.

```
void CGEventKeyboardGetUnicodeString (
   CGEventRef event,
   UniCharCount maxStringLength,
   UniCharCount *actualStringLength,
   UniChar unicodeString[]
);
```

Parameters

event

The keyboard event to access.

maxStringLength

The length of the array you provide in the `unicodeString` parameter.

actualStringLength

A pointer to a `UniCharCount` variable. On return, the variable contains the actual count of Unicode characters in the event data.

unicodeString

A pointer to a `UniChar` array. You are responsible for allocating storage for the array. On return, your array contains the Unicode string associated with the specified event.

Discussion
When you call this function and specify a `NULL` string or a maximum string length of 0, the function still returns the actual count of Unicode characters in the event data.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventKeyboardSetUnicodeString

Sets the Unicode string associated with a Quartz keyboard event.

```
void CGEventKeyboardSetUnicodeString (
   CGEventRef event,
   UniCharCount stringLength,
   const UniChar unicodeString[]
);
```

Parameters

event
The keyboard event to access.

stringLength
The length of the array you provide in the `unicodeString` parameter.

unicodeString
An array that contains the new Unicode string associated with the specified event.

Discussion
By default, the system translates the virtual key code in a keyboard event into a Unicode string based on the keyboard ID in the event source. This function allows you to manually override this string. Note that application frameworks may ignore the Unicode string in a keyboard event and do their own translation based on the virtual keycode and perceived event state.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventMaskBit

Generates an event mask for a single type of event.

```
CGEventMask CGEventMaskBit (
   CGEventType eventType
);
```

Parameters

eventType
An event type constant. Pass one of the constants listed in "Event Types" (page 1623).

Return Value
An event mask that represents the specified event.

Discussion
This macro converts an event type constant into a mask. You can use this mask to specify that an event tap should observe the event. For more information, see `CGEventMask` (page 1606).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

CGEventPost

Posts a Quartz event into the event stream at a specified location.

```
void CGEventPost (
   CGEventTapLocation tap,
   CGEventRef event
);
```

Parameters

tap
The location at which to post the event. Pass one of the constants listed in "Event Tap Locations" (page 1621).

event
The event to post.

Discussion
This function posts the specified event immediately before any event taps instantiated for that location, and the event passes through any such taps.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventPostToPSN

Posts a Quartz event into the event stream for a specific application.

```
void CGEventPostToPSN (
   void *processSerialNumber,
   CGEventRef event
);
```

Parameters

processSerialNumber
The process to receive the event.

event
The event to post.

Discussion
This function makes it possible for an application to establish an event routing policy, for example, by tapping events at the `kCGAnnotatedSessionEventTap` location and then posting the events to another desired process.

This function posts the specified event immediately before any event taps instantiated for the specified process, and the event passes through any such taps.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventSetDoubleValueField

Sets the floating-point value of a field in a Quartz event.

```
void CGEventSetDoubleValueField (
   CGEventRef event,
   CGEventField field,
   double value
);
```

Parameters

event
The event to access.

field
A field in the specified event. Pass one of the constants listed in "Event Fields" (page 1610).

value
The new value of the specified field.

Discussion
Before calling this function, the event type must be set using a typed event creation function such as `CGEventCreateMouseEvent` (page 1574), or by calling `CGEventSetType` (page 1586).

In cases where the field's value is represented within the event by a fixed point number or integer, the `value` parameter is scaled as needed and converted to the appropriate type.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventSetFlags

Sets the event flags of a Quartz event.

```
void CGEventSetFlags (
   CGEventRef event,
   CGEventFlags flags
);
```

Parameters

event

The event to access.

location

The new flags of the specified event. See "Event Flags" (page 1618).

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventSetIntegerValueField

Sets the integer value of a field in a Quartz event.

```
void CGEventSetIntegerValueField (
   CGEventRef event,
   CGEventField field,
   int64_t value
);
```

Parameters

event

The event to access.

field

A field in the specified event. Pass one of the constants listed in "Event Fields" (page 1610).

value

The new value of the specified field.

Discussion
Before calling this function, the event type must be set using a typed event creation function such as `CGEventCreateMouseEvent` (page 1574), or by calling `CGEventSetType` (page 1586).

If you are creating a mouse event generated by a tablet, call this function and specify the field `kCGMouseEventSubtype` with a value of `kCGEventMouseSubtypeTabletPoint` or `kCGEventMouseSubtypeTabletProximity` before setting other parameters.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventSetLocation

Sets the location of a Quartz mouse event.

```
void CGEventSetLocation (
   CGEventRef event,
   CGPoint location
);
```

Parameters

event

The mouse event whose location to set.

location

The new location of the specified mouse event in global display coordinates.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventSetSource

Sets the event source of a Quartz event.

```
void CGEventSetSource (
   CGEventRef event,
   CGEventSourceRef source
);
```

Parameters

event

The event to access.

source

The new event source of the specified event.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventSetTimestamp

Sets the timestamp of a Quartz event.

```
void CGEventSetTimestamp (
   CGEventRef event,
   CGEventTimestamp timestamp
);
```

Parameters

event

The event to access.

timestamp

The new timestamp of the specified event.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventSetType

Sets the event type of a Quartz event (left mouse down, for example).

```
void CGEventSetType (
   CGEventRef event,
   CGEventType type
);
```

Parameters

event
The event to access.

type
The new event type of the specified event. The return value is one of the constants listed in "Event Types" (page 1623).

Availability
Available in Mac OS X v10.4 and later.

See Also
`CGEventGetType` (page 1579)

Declared In
`CGEvent.h`

CGEventSourceButtonState

Returns a Boolean value indicating the current button state of a Quartz event source.

```
bool CGEventSourceButtonState (
   CGEventSourceStateID sourceState,
   CGMouseButton button
);
```

Parameters

sourceState
The source state to access. Pass one of the constants listed in "Event Source States" (page 1619).

button
The mouse button to test. Pass one of the constants listed in "Mouse Buttons" (page 1626).

Return Value
If `true`, the button is down. If `false`, the button is up.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceCounterForEventType

Returns a count of events of a given type seen since the window server started.

```
uint32_t CGEventSourceCounterForEventType (
   CGEventSourceStateID source,
   CGEventType evType
);
```

Parameters

sourceState
The source state to access. Pass one of the constants listed in "Event Source States" (page 1619).

eventType
The event type to access. To get the count of input events—keyboard, mouse, or tablet—specify kCGAnyInputEventType.

Return Value
The count of events of the specified type seen since the window server started.

Discussion
Quartz provides these counters for applications that monitor user activity. For example, an application could prompt a typist to take a break to reduce repetitive stress injuries.

Modifier keys produce kCGEventFlagsChanged events, not kCGEventKeyDown events, and do so both on press and release. The volume, brightness, and CD eject keys on some keyboards (both desktop and laptop) do not generate key up or key down events.

For various reasons, the number of key up and key down events may not be the same when all keyboard keys are up. As a result, a mismatch does not necessarily indicate that some keys are down.

Key autorepeat events are not counted.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceCreate

Returns a Quartz event source created with a specified source state.

```
CGEventSourceRef CGEventSourceCreate (
   CGEventSourceStateID sourceState
);
```

Parameters

sourceState
The event state table to use for this event source. Pass one of the constants listed in "Event Source States" (page 1619).

Return Value
A new event source, or `NULL` if the specified source state is not valid. When you no longer need the event source, you should release it using the function `CFRelease`.

Discussion
If two or more event sources are using the same source state and one of them is released, the remaining event sources will behave as if all keys and buttons on input devices are up in generating new events from this source.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventSource.h`

CGEventSourceFlagsState

Returns the current flags of a Quartz event source.

```
CGEventFlags CGEventSourceFlagsState (
   CGEventSourceStateID sourceState
);
```

Parameters

sourceState
The source state to access. Pass one of the constants listed in "Event Source States" (page 1619).

Return Value
The current flags of the specified event source. For more information, see "Event Flags" (page 1618).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventSource.h`

CGEventSourceGetKeyboardType

Returns the keyboard type to be used with a Quartz event source.

```
CGEventSourceKeyboardType CGEventSourceGetKeyboardType (
   CGEventSourceRef source
);
```

Parameters

source
The event source to access. Pass one of the constants listed in "Event Source States" (page 1619).

Return Value
The keyboard type to be used with the specified event source.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceGetLocalEventsFilterDuringSuppressionState

Returns the mask that indicates which classes of local hardware events are enabled during event suppression.

```
CGEventFilterMask CGEventSourceGetLocalEventsFilterDuringSuppressionState (
   CGEventSourceRef source,
   CGEventSuppressionState state
);
```

Parameters

source

The event source to access.

state

The type of event suppression interval during which the filter is applied. Pass one of the constants listed in "Event Suppression States" (page 1621).

Return Value
A mask that specifies the categories of local hardware events to enable during the event suppression interval. See "Event Filter Masks" (page 1618).

Discussion
You can configure the system to suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted or during a synthetic mouse drag (mouse movement with the left or only mouse button down). For information about setting this local events filter, see `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 1593).

This function lets you specify an event source and a suppression state (event suppression interval or mouse drag), and returns a filter mask of event categories to be passed through during suppression.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceGetLocalEventsSuppressionInterval

Returns the interval that local hardware events may be suppressed following the posting of a Quartz event.

```
CFTimeInterval CGEventSourceGetLocalEventsSuppressionInterval (
   CGEventSourceRef source
);
```

Parameters

source

The event source to access.

Discussion
By default, the system does not suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted. You can use the function `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 1593) to modify this behavior.

This function gets the period of time in seconds that local hardware events may be suppressed after posting a Quartz event created with the specified event source. You can use the function `CGEventSourceSetLocalEventsSuppressionInterval` (page 1594) to change this time interval.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventSource.h`

CGEventSourceGetPixelsPerLine

Gets the scale of pixels per line in a scrolling event source.

```
double CGEventSourceGetPixelsPerLine (
    CGEventSourceRef source
);
```

Parameters

source
The event source to access.

Return Value
The scale of pixels per line in a scrolling event.

Discussion
This function returns the scale of pixels per line in the specified event source. For example, if the scale in the event source is 10.5 pixels per line, this function would return 10.5. Every scrolling event can be interpreted to be scrolling by pixel or by line. By default, the scale is about ten pixels per line. You can alter the scale with the function `CGEventSourceSetPixelsPerLine`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`CGEventSourceSetPixelsPerLine` (page 1594)

Declared In
`CGEventSource.h`

CGEventSourceGetSourceStateID

Returns the source state associated with a Quartz event source.

```
CGEventSourceStateID CGEventSourceGetSourceStateID (
   CGEventSourceRef source
);
```

Parameters

source
The event source to access.

Return Value
The source state associated with the specified event source.

Discussion
This function returns the ID of the source state table associated with an event source.

For event sources created with the `kCGEventSourceStatePrivate` source state, this function returns the ID of the private source state table created for the event source. This unique ID may be passed to the `CGEventSourceCreate` function to create a second event source sharing the same state table. This may be useful, for example, in creating separate mouse and keyboard sources which share a common private state.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventSource.h`

CGEventSourceGetTypeID

Returns the type identifier for the opaque type `CGEventSourceRef`.

```
CFTypeID CGEventSourceGetTypeID (
   void
);
```

Return Value
The Core Foundation type identifier for the opaque type `CGEventSourceRef` (page 1607).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventSource.h`

CGEventSourceGetUserData

Returns the 64-bit user-specified data for a Quartz event source.

```
int64_t CGEventSourceGetUserData (
   CGEventSourceRef source
);
```

Parameters

source
The event source to access.

Return Value
The user-specified data.

Discussion
Each input event includes 64 bits of user-specified data. This function gets the user-specified data for all events created by the specified event source. This data may also be obtained per event using the `CGEventGetIntegerValueField` (page 1577) function.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceKeyState

Returns a Boolean value indicating the current keyboard state of a Quartz event source.

```
bool CGEventSourceKeyState (
   CGEventSourceStateID sourceState,
   CGKeyCode key
);
```

Parameters

sourceState
The source state to access. Pass one of the constants listed in "Event Source States" (page 1619).

key
The virtual key code to test.

Return Value
If `true`, the key is down. If `false`, the key is up.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceSecondsSinceLastEventType

Returns the elapsed time since the last event for a Quartz event source.

```
CFTimeInterval CGEventSourceSecondsSinceLastEventType (
   CGEventSourceStateID source,
   CGEventType eventType
);
```

Parameters

source
The source state to access. Pass one of the constants listed in "Event Source States" (page 1619).

eventType
The event type to access. To get the elapsed time since the previous input event—keyboard, mouse, or tablet—specify `kCGAnyInputEventType`.

Return Value
The time in seconds since the previous input event of the specified type.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceSetKeyboardType

Sets the keyboard type to be used with a Quartz event source.

```
void CGEventSourceSetKeyboardType (
   CGEventSourceRef source,
   CGEventSourceKeyboardType keyboardType
);
```

Parameters

source
The event source to access.

keyboardType
The keyboard type to be used with the specified event source.

Availability

Available in Mac OS X v10.4 and later.

Declared In

CGEventSource.h

CGEventSourceSetLocalEventsFilterDuringSuppressionState

Sets the mask that indicates which classes of local hardware events are enabled during event suppression.

```
void CGEventSourceSetLocalEventsFilterDuringSuppressionState (
   CGEventSourceRef source,
   CGEventFilterMask filter,
   CGEventSuppressionState state
);
```

Parameters

source
The event source to access.

filter
A mask that specifies the categories of local hardware events to enable during the event suppression interval. See "Event Filter Masks" (page 1618).

state
The type of event suppression interval during which the filter is applied. Pass one of the constants listed in "Event Suppression States" (page 1621).

Discussion

By default, the system does not suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted—see `CGEventSourceSetLocalEventsSuppressionInterval` (page 1594)—and during a synthetic mouse drag (mouse movement with the left or only mouse button down).

Some applications may want to disable events from some of the local hardware during this interval. For example, if you post mouse events only, you may wish to suppress local mouse events and permit local keyboard events to pass through. This function lets you specify an event source, a suppression state (event suppression interval or mouse drag), and a filter mask of event classes to be passed through. The new local events filter takes effect with the next Quartz event you post using this event source.

Availability

Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceSetLocalEventsSuppressionInterval

Sets the interval that local hardware events may be suppressed following the posting of a Quartz event.

```
void CGEventSourceSetLocalEventsSuppressionInterval (
   CGEventSourceRef source,
   CFTimeInterval seconds
);
```

Parameters

source
The event source to access.

seconds
The period of time in seconds that local hardware events (keyboard or mouse) are suppressed after posting a Quartz event created with the specified event source. The value should be a number in the range [0.0, 10.0].

Discussion
By default, the system does not suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted. You can use the function CGEventSourceSetLocalEventsFilterDuringSuppressionState (page 1593) to modify this behavior.

This function sets the period of time in seconds that local hardware events may be suppressed after posting a Quartz event created with the specified event source. The default suppression interval is 0.25 seconds.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEventSource.h

CGEventSourceSetPixelsPerLine

Sets the scale of pixels per line in a scrolling event source.

```
void CGEventSourceSetPixelsPerLine (
    CGEventSourceRef source,
    double pixelsPerLine
);
```

Parameters

source
The event source to access.

pixelsPerLine
The scale of pixels per line in the specified event source.

Discussion
This function sets the scale of pixels per line in the specified event source. For example, if you pass the value 12.0 in the *pixelsPerLine* parameter, the scale of pixels per line in the event source would be changed to 12.0. Every scrolling event can be interpreted to be scrolling by pixel or by line. By default, the scale is about ten pixels per line. You can retrieve the scale with the function `CGEventSourceGetPixelsPerLine`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`CGEventSourceGetPixelsPerLine` (page 1590)

Declared In
`CGEventSource.h`

CGEventSourceSetUserData

Sets the 64-bit user-specified data for a Quartz event source.

```
void CGEventSourceSetUserData (
   CGEventSourceRef source,
   int64_t userData
);
```

Parameters

source
The event source to access.

userData
The user-specified data. For example, you could specify a vendor hardware ID.

Discussion
Each input event includes 64 bits of user-specified data. This function sets the user-specified data for all events created by the specified event source. This data may also be set per event using the `CGEventGetIntegerValueField` (page 1577) function.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventSource.h`

CGEventTapCreate

Creates an event tap.

```
CFMachPortRef CGEventTapCreate (
   CGEventTapLocation tap,
   CGEventTapPlacement place,
   CGEventTapOptions options,
   CGEventMask eventsOfInterest,
   CGEventTapCallBack callback,
   void *refcon
);
```

Parameters

tap

The location of the new event tap. Pass one of the constants listed in "Event Tap Locations" (page 1621). Only processes running as the root user may locate an event tap at the point where HID events enter the window server; for other users, this function returns `NULL`.

place

The placement of the new event tap in the list of active event taps. Pass one of the constants listed in "Event Tap Placement" (page 1622).

options

A constant that specifies whether the new event tap is a passive listener or an active filter.

eventsOfInterest

A bit mask that specifies the set of events to be observed. For a list of possible events, see "Event Types" (page 1623). For information on how to specify the mask, see `CGEventMask` (page 1606). If the event tap is not permitted to monitor one or more of the events specified in the `eventsOfInterest` parameter, then the appropriate bits in the mask are cleared. If that action results in an empty mask, this function returns `NULL`.

callback

An event tap callback function that you provide. Your callback function is invoked from the run loop to which the event tap is added as a source. The thread safety of the callback is defined by the run loop's environment. To learn more about event tap callbacks, see `CGEventTapCallBack` (page 1604).

refcon

A pointer to user-defined data. This pointer is passed into the callback function specified in the `callback` parameter.

Return Value

A Core Foundation mach port that represents the new event tap, or `NULL` if the event tap could not be created. When you are finished using the event tap, you should release the mach port using the function `CFRelease`. Releasing the mach port also releases the tap.

Discussion

Event taps receive key up and key down events if one of the following conditions is true:

- The current process is running as the root user.
- Access for assistive devices is enabled. In Mac OS X v10.4, you can enable this feature using System Preferences, Universal Access panel, Keyboard view.

After creating an event tap, you can add it to a run loop as follows:

1. Pass the event tap to the `CFMachPortCreateRunLoopSource` function to create a run loop event source.
2. Call the `CFRunLoopAddSource` function to add the source to the appropriate run loop.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGEventTapCreateForPSN

Creates an event tap for a specified process.

```
CFMachPortRef CGEventTapCreateForPSN (
   void *processSerialNumber,
   CGEventTapPlacement place,
   CGEventTapOptions options,
   CGEventMask eventsOfInterest,
   CGEventTapCallBack callback,
   void *refcon
);
```

Parameters

processSerialNumber
The process to monitor.

place
The placement of the new event tap in the list of active event taps. Pass one of the constants listed in "Event Tap Placement" (page 1622).

options
A constant that specifies whether the new event tap is a passive listener or an active filter.

eventsOfInterest
A bit mask that specifies the set of events to be observed. For a list of possible events, see "Event Types" (page 1623). For information on how to specify the mask, see `CGEventMask` (page 1606). If the event tap is not permitted to monitor one or more of the events specified in the `eventsOfInterest` parameter, then the appropriate bits in the mask are cleared. If that action results in an empty mask, this function returns `NULL`.

callback
An event tap callback function that you provide. Your callback function is invoked from the run loop to which the event tap is added as a source. The thread safety of the callback is defined by the run loop's environment. To learn more about event tap callbacks, see `CGEventTapCallBack` (page 1604).

refcon
A pointer to user-defined data. This pointer is passed into the callback function specified in the `callback` parameter.

Return Value
A Core Foundation mach port that represents the new event tap, or `NULL` if the event tap could not be created. When you are finished using the event tap, you should release the mach port using the function `CFRelease`. Releasing the mach port also releases the tap.

Discussion
This function creates an event tap that receives events being routed by the window server to the specified process. For more information about creating event taps, see `CGEventTapCreate` (page 1595).

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventTapEnable

Enables or disables an event tap.

```
void CGEventTapEnable (
   CFMachPortRef myTap,
   bool enable
);
```

Parameters

myTap

The event tap to enable or disable.

enable

Pass `true` to enable the event tap. To disable it, pass `false`.

Discussion
Event taps are normally enabled when created. If an event tap becomes unresponsive, or if a user requests that event taps be disabled, then a `kCGEventTapDisabled` event is passed to the event tap callback function. Event taps may be re-enabled by calling this function.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventTapIsEnabled

Returns a Boolean value indicating whether an event tap is enabled.

```
bool CGEventTapIsEnabled (
   CFMachPortRef myTap
);
```

Parameters

myTap

The event tap to test.

Return Value
If `true`, the specified event tap is enabled; otherwise, `false`.

Discussion
For more information, see the function `CGEventTapEnable` (page 1598).

Availability
Available in Mac OS X v10.4 and later.

Declared In
CGEvent.h

CGEventTapPostEvent

Posts a Quartz event from an event tap into the event stream.

```
void CGEventTapPostEvent (
   CGEventTapProxy proxy,
   CGEventRef event
);
```

Parameters

proxy

A proxy that identifies the event tap posting the event. Your event tap callback function is passed this proxy when it is invoked.

event

The event to post.

Discussion

You can use this function to post a new event at the same point to which an event returned from an event tap callback function would be posted. The new event enters the system before the event returned by the callback enters the system. Events posted into the system will be seen by all taps placed after the tap posting the event.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGGetEventTapList

Gets a list of currently installed event taps.

```
CGError CGGetEventTapList (
   CGTableCount maxNumberOfTaps,
   CGEventTapInformation tapList[],
   CGTableCount *eventTapCount
);
```

Parameters

maxNumberOfTaps

The length of the array you provide in the `tapList` parameter.

tapList

An array of event tap information structures. You are responsible for allocating storage for the array. On return, your array contains a list of currently installed event taps. If you pass `NULL` in this parameter, the `maxNumberOfTaps` parameter is ignored, and the `eventTapCount` variable is filled in with the number of event taps that are currently installed.

eventTapCount

A pointer to a `CGTableCount` variable. On return, the variable contains actual number of array elements filled in.

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion
Each call to this function has the side effect of resetting the minimum and maximum latency values in the `tapList` parameter to the corresponding average values. Values reported in these fields reflect the minimum and maximum values seen since the preceding call, or the instantiation of the tap. This allows a monitoring tool to evaluate the best and worst case latency over time and under various operating conditions.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEvent.h`

CGInhibitLocalEvents

Turns off local hardware events in the current session.

```
CGError CGInhibitLocalEvents (
   boolean_t doInhibit
);
```

Parameters

`doInhibit`
Pass `true` to specify that local hardware events on the remote system should be inhibited; otherwise, pass `false`.

Return Value
A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion
This function is typically used during remote operation of a system to disconnect the keyboard and mouse for a short period of time, as in automated system testing or telecommuting applications.

The `CGInhibitLocalEvents` function is not recommended for general use because of undocumented special cases and undesirable side effects. For example, this function can permanently disable the keyboard and mouse, rendering the system unusable. The recommended replacement for this function is `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 1593).

Special Considerations

In Mac OS X v10.2 and earlier, this function inhibits local events only after a synthetic keyboard or mouse event is posted by the calling application. In Mac OS X v10.3 and later, event inhibition takes effect immediately. If your application terminates for any reason, event inhibition on the remote system is immediately turned off.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGRemoteOperation.h`

CGPostKeyboardEvent

Synthesizes a low-level keyboard event on the local machine.

```
CGError CGPostKeyboardEvent (
   CGCharCode keyChar,
   CGKeyCode virtualKey,
   boolean_t keyDown
);
```

Parameters

keyChar

The value of the character to generate, or 0 to specify that the system should guess an appropriate value based on the default key mapping.

virtualKey

The virtual key code for the event. See `CGKeyCode` (page 1609).

keyDown

Pass `true` to specify that the key position is down; otherwise, pass `false`.

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventCreateKeyboardEvent` (page 1573), which allows you to create a keyboard event and customize the event before posting it to the event system.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGPostMouseEvent

Synthesizes a low-level mouse-button event on the local machine.

```
CGError CGPostMouseEvent (
   CGPoint mouseCursorPosition,
   boolean_t updateMouseCursorPosition,
   CGButtonCount buttonCount,
   boolean_t mouseButtonDown,
   ...
);
```

Parameters

mouseCursorPosition

The new coordinates of the mouse in global display space.

updateMouseCursorPosition

Pass `true` if the on-screen cursor should be moved to the location specified in the `mouseCursorPosition` parameter; otherwise, pass `false`.

buttonCount

The number of mouse buttons, up to a maximum of 32.

mouseButtonDown

Pass `true` to specify that the primary or left mouse button is down; otherwise, pass `false`.

`...`
Zero or more Boolean values that specify whether the remaining mouse buttons are down (`true`) or up (`false`). The second value, if any, should specify the state of the secondary mouse button (right). A third value would specify the state of the center button, and the remaining buttons would be in USB device order.

Return Value
A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion
Based on the arguments you pass to this function, the function generates the appropriate mouse-down, mouse-up, mouse-move, or mouse-drag events by comparing the new state with the current state.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventCreateMouseEvent` (page 1574), which allows you to create a mouse event and customize the event before posting it to the event system.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGRemoteOperation.h`

CGPostScrollWheelEvent

Synthesizes a low-level scrolling event on the local machine.

```
CGError CGPostScrollWheelEvent (
   CGWheelCount wheelCount,
   int32_t wheel1,
   ...
);
```

Parameters

`wheelCount`
The number of scrolling devices, up to a maximum of 3.

`wheel1`
A value that reflects the movement of the primary scrolling device on the mouse.

`...`
Up to two values that reflect the movements of the other scrolling devices on the mouse (if any).

Return Value
A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion
Scrolling movement is generally represented by small signed integer values, typically in a range from -10 to +10. Large values may have unexpected results, depending on the application that processes the event.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventCreateScrollWheelEvent` (page 1575), which allows you to create a scrolling event and customize the event before posting it to the event system.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGRemoteOperation.h

CGSetLocalEventsFilterDuringSuppressionState

Filters local hardware events from the keyboard and mouse during the short interval after a synthetic event is posted.

```
CGError CGSetLocalEventsFilterDuringSuppressionState (
   CGEventFilterMask filter,
   CGEventSuppressionState state
);
```

Parameters

filter

The class of local hardware events to enable after a synthetic event is posted. Pass one of the constants listed in "Event Filter Masks" (page 1618).

state

The type of interval during which the filter is applied. Pass one of the constants listed in "Event Suppression States" (page 1621).

Return Value
A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion
By default, the system suppresses local hardware events from the keyboard and mouse during a short interval after a synthetic event is posted and during a synthetic mouse drag (mouse movement with the left or only mouse button down).

Some applications may want to enable events from some of the local hardware. For example, if you post mouse events only, you may wish to permit local keyboard hardware events to pass through.

This function lets you specify a state (event suppression interval or mouse drag), and a mask of event categories to be passed through. The new filter state takes effect with the next synthetic event you post.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 1593), which allows the filter behavior to be associated only with events created from a specific event source.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CGRemoteOperation.h

CGSetLocalEventsSuppressionInterval

Sets the time interval in seconds that local hardware events are suppressed after posting a synthetic event.

```
CGError CGSetLocalEventsSuppressionInterval (
   CFTimeInterval seconds
);
```

Parameters

seconds

The desired time interval in seconds. The value should be a number in the range [0.0, 10.0].

Return Value

A result code. If the `seconds` parameter is outside the allowed range, returns `kCGErrorRangeCheck`.

Discussion

This function determines how long local events matching an event filter are to be suppressed following the posting of a synthetic event. The default time interval for event suppression is 0.25 seconds.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventSourceSetLocalEventsSuppressionInterval` (page 1594), which allows the suppression interval to be adjusted for a specific event source, affecting only events posted using that event source.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

Callbacks

CGEventTapCallBack

A client-supplied callback function that's invoked whenever an associated event tap receives a Quartz event.

```
typedef CGEventRef (*CGEventTapCallBack) (
   CGEventTapProxy proxy,
   CGEventType type,
   CGEventRef event,
   void *refcon
);
```

If you name your function `MyEventTapCallBack`, you would declare it like this:

```
CGEventRef MyEventTapCallBack (
   CGEventTapProxy proxy,
   CGEventType type,
   CGEventRef event,
   void *refcon
);
```

Parameters

proxy

A proxy for the event tap. See `CGEventTapProxy` (page 1609). This callback function may pass this proxy to other functions such as the event-posting routines.

type
The event type of this event. See "Event Types" (page 1623).

event
The incoming event. This event is owned by the caller, and you do not need to release it.

refcon
A pointer to user-defined data. You specify this pointer when you create the event tap. Several different event taps could use the same callback function, each tap with its own user-defined data.

Discussion
If the event tap is an active filter, your callback function should return one of the following:

- The (possibly modified) event that is passed in. This event is passed back to the event system.
- A newly-constructed event. After the new event has been passed back to the event system, the new event will be released along with the original event.
- `NULL` if the event passed in is to be deleted.

If the event tap is an passive listener, your callback function may return the event that is passed in, or `NULL`. In either case, the event stream is not affected.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

Data Types

CGButtonCount

Represents the number of buttons being set in a synthetic mouse event.

```
typedef uint32_t CGButtonCount;
```

Discussion
In mouse events, the button count parameter ranges from 0 to 31. See the function `CGPostMouseEvent` (page 1601).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGRemoteOperation.h`

CGCharCode

Represents a character generated by pressing one or more keys on a keyboard.

```
typedef uint16_t CGCharCode;
```

Discussion
This data type represents a 16-bit character code. Values of this type may or may not correspond to UTF-16 character codes. See the function `CGPostKeyboardEvent` (page 1600).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGRemoteOperation.h`

CGEventMask

Defines a mask that identifies the set of Quartz events to be observed in an event tap.

```
typedef uint64_t CGEventMask;
```

Discussion
When you call either `CGEventTapCreate` (page 1595) or `CGEventTapCreateForPSN` (page 1597) to register an event tap, you supply a bit mask that identifies the set of events to be observed. You specify each event using one of the event type constants listed in "Event Types" (page 1623). To form the bit mask, use the `CGEventMaskBit` macro to convert each constant into an event mask and then OR the individual masks together. For example:

```
CGEventMask mask = CGEventMaskBit(kCGEventLeftMouseDown) |
                   CGEventMaskBit(kCGEventLeftMouseUp);
```

You can also supply a mask to observe all events:

```
CGEventMask mask = kCGEventMaskForAllEvents;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

CGEventRef

Defines an opaque type that represents a low-level hardware event.

```
typedef struct __CGEvent *CGEventRef;
```

Discussion
Low-level hardware events of this type are referred to as Quartz events. A typical event in Mac OS X originates when the user manipulates an input device such as a mouse or a keyboard. The device driver associated with that device, through the I/O Kit, creates a low-level event, puts it in the window server's event queue, and notifies the window server. The window server creates a Quartz event, annotates the event, and dispatches the event to the appropriate run-loop port of the target process. There the event is picked up by the Carbon Event Manager and forwarded to the event-handling mechanism appropriate to the application environment. You can use event taps to gain access to Quartz events at several different steps in this process.

This opaque type is derived from CFType and inherits the properties that all Core Foundation types have in common. For more information, see *CFType Reference*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

CGEventSourceKeyboardType

Defines a code that represents the type of keyboard used with a specified event source.

```
typedef uint32_t CGEventSourceKeyboardType;
```

Discussion
This code is the same keyboard type identifier used with the `UCKeyTranslate` function to drive keyboard translation.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

CGEventSourceRef

Defines an opaque type that represents the source of a Quartz event.

```
typedef struct __CGEventSource * CGEventSourceRef;
```

Discussion
A Quartz event source is an object that contains accumulated state related to event generation and event posting. Every event source has an associated global event state table called a source state. When you call `CGEventSourceCreate` (page 1587) to create an event source, you specify which source state to use. For more information about source states, see "Event Source States" (page 1619).

A typical use of an event source would be to obtain the source from a Quartz event received by an event tap callback function, and then to use that source for any new events created as a result of the received event. This has the effect of marking the events as being related.

This opaque type is derived from CFType and inherits the properties that all Core Foundation types have in common. For more information, see *CFType Reference*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

CGEventTapInformation

Defines the structure used to report information about event taps.

```
typedef struct CGEventTapInformation
{
    uint32_t            eventTapID;
    CGEventTapLocation  tapPoint;
    CGEventTapOptions   options;
    CGEventMask         eventsOfInterest;
    pid_t               tappingProcess;
    pid_t               processBeingTapped;
    bool                enabled;
    float               minUsecLatency;
    float               avgUsecLatency;
    float               maxUsecLatency;
} CGEventTapInformation;
```

Fields

`eventTapID`
The unique identifier for the event tap.

`tapPoint`
The location of the event tap. See "Event Tap Locations" (page 1621).

`options`
The type of event tap (passive listener or active filter).

`eventsOfInterest`
The mask that identifies the set of events to be observed.

`tappingProcess`
The process ID of the application that created the event tap.

`processBeingTapped`
The process ID of the target application (non-zero only if the event tap was created using the function `CGEventTapCreateForPSN` (page 1597).

`enabled`
`TRUE` if the event tap is currently enabled; otherwise `FALSE`.

`minUsecLatency`
Minimum latency in microseconds. In this data structure, **latency** is defined as the time in microseconds it takes for an event tap to process and respond to an event passed to it.

`avgUsecLatency`
Average latency in microseconds. This is a weighted average that gives greater weight to more recent events.

`maxUsecLatency`
Maximum latency in microseconds.

Discussion

To learn how to obtain information about event taps, see the function `CGGetEventTapList` (page 1599).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventTypes.h`

CGEventTapProxy

Defines an opaque type that represents state within the client application that's associated with an event tap.

```
typedef struct __CGEventTapProxy * CGEventTapProxy;
```

Discussion
An event tap proxy object is passed to your event tap callback function when it receives a new Quartz event. Your callback function needs the proxy to post Quartz events using the function `CGEventTapPostEvent` (page 1599).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

CGEventTimestamp

Defines the elapsed time in nanoseconds since startup that a Quartz event occurred.

```
typedef uint64_t CGEventTimestamp;
```

Discussion
An event timestamp is a big, unsigned, 64-bit number. That's big, really big. You just won't believe how vastly, hugely, mind-bogglingly big it is. You may think your application has been running for a long time, but that's just peanuts to an event timestamp.

For information about how event timestamps are used, see the functions `CGEventGetTimestamp` (page 1578) and `CGEventSetTimestamp` (page 1585).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGEventTypes.h`

CGKeyCode

Represents the virtual key codes used in keyboard events.

```
typedef uint16_t CGKeyCode;
```

Discussion
In Mac OS X, the hardware scan codes generated by keyboards are mapped to a set of virtual key codes that are hardware-independent. Pressing a given key always generates the same virtual key code on any supported keyboard.

As keys are pressed, the system uses the virtual key codes to create low-level keyboard events. For information on how to simulate a keyboard event, see the function `CGEventCreateKeyboardEvent` (page 1573).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGRemoteOperation.h

CGWheelCount

Represents the number of wheels being set in a scroll wheel event.

```
typedef uint32_t CGWheelCount;
```

Discussion
See the function CGPostScrollWheelEvent (page 1602).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CGRemoteOperation.h

Constants

Event Fields

Constants used as keys to access specialized fields in low-level events.

```
enum _CGEventField {
    kCGMouseEventNumber = 0,
    kCGMouseEventClickState = 1,
    kCGMouseEventPressure = 2,
    kCGMouseEventButtonNumber = 3,
    kCGMouseEventDeltaX = 4,
    kCGMouseEventDeltaY = 5,
    kCGMouseEventInstantMouser = 6,
    kCGMouseEventSubtype = 7,
    kCGKeyboardEventAutorepeat = 8,
    kCGKeyboardEventKeycode = 9,
    kCGKeyboardEventKeyboardType = 10,
    kCGScrollWheelEventDeltaAxis1 = 11,
    kCGScrollWheelEventDeltaAxis2 = 12,
    kCGScrollWheelEventDeltaAxis3 = 13,
    kCGScrollWheelEventFixedPtDeltaAxis1 = 93,
    kCGScrollWheelEventFixedPtDeltaAxis2 = 94,
    kCGScrollWheelEventFixedPtDeltaAxis3 = 95,
    kCGScrollWheelEventPointDeltaAxis1 = 96,
    kCGScrollWheelEventPointDeltaAxis2 = 97,
    kCGScrollWheelEventPointDeltaAxis3 = 98,
    kCGScrollWheelEventInstantMouser = 14,
    kCGTabletEventPointX = 15,
    kCGTabletEventPointY = 16,
    kCGTabletEventPointZ = 17,
    kCGTabletEventPointButtons = 18,
    kCGTabletEventPointPressure = 19,
    kCGTabletEventTiltX = 20,
    kCGTabletEventTiltY = 21,
    kCGTabletEventRotation = 22,
    kCGTabletEventTangentialPressure = 23,
    kCGTabletEventDeviceID = 24,
    kCGTabletEventVendor1 = 25,
    kCGTabletEventVendor2 = 26,
    kCGTabletEventVendor3 = 27,
    kCGTabletProximityEventVendorID = 28,
    kCGTabletProximityEventTabletID = 29,
    kCGTabletProximityEventPointerID = 30,
    kCGTabletProximityEventDeviceID = 31,
    kCGTabletProximityEventSystemTabletID = 32,
    kCGTabletProximityEventVendorPointerType = 33,
    kCGTabletProximityEventVendorPointerSerialNumber = 34,
    kCGTabletProximityEventVendorUniqueID = 35,
    kCGTabletProximityEventCapabilityMask = 36,
    kCGTabletProximityEventPointerType = 37,
    kCGTabletProximityEventEnterProximity = 38,
    kCGEventTargetProcessSerialNumber = 39,
    kCGEventTargetUnixProcessID = 40,
    kCGEventSourceUnixProcessID = 41,
    kCGEventSourceUserData = 42,
    kCGEventSourceUserID = 43,
    kCGEventSourceGroupID = 44,
    kCGEventSourceStateID = 45,
    kCGScrollWheelEventIsContinuous = 88
};
typedef uint32_t CGEventField;
```

Constants

`kCGMouseEventNumber`
Key to access an integer field that contains the mouse button event number. Matching mouse-down and mouse-up events will have the same event number.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseEventClickState`
Key to access an integer field that contains the mouse button click state. A click state of 1 represents a single click. A click state of 2 represents a double-click. A click state of 3 represents a triple-click.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseEventPressure`
Key to access a double field that contains the mouse button pressure. The pressure value may range from 0 to 1, with 0 representing the mouse being up. This value is commonly set by tablet pens mimicking a mouse.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseEventButtonNumber`
Key to access an integer field that contains the mouse button number. For information about the possible values, see "Mouse Buttons" (page 1626).

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseEventDeltaX`
Key to access an integer field that contains the horizontal mouse delta since the last mouse movement event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseEventDeltaY`
Key to access an integer field that contains the vertical mouse delta since the last mouse movement event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseEventInstantMouser`
Key to access an integer field. The value is non-zero if the event should be ignored by the Inkwell subsystem.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseEventSubtype`
Key to access an integer field that encodes the mouse event subtype as a `kCFNumberIntType`.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGKeyboardEventAutorepeat`
Key to access an integer field, non-zero when this is an autorepeat of a key-down, and zero otherwise.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGKeyboardEventKeycode`
: Key to access an integer field that contains the virtual keycode of the key-down or key-up event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGKeyboardEventKeyboardType`
: Key to access an integer field that contains the keyboard type identifier.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventDeltaAxis1`
: Key to access an integer field that contains scrolling data. This field typically contains the change in vertical position since the last scrolling event from a Mighty Mouse scroller or a single-wheel mouse scroller.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventDeltaAxis2`
: Key to access an integer field that contains scrolling data. This field typically contains the change in horizontal position since the last scrolling event from a Mighty Mouse scroller.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventDeltaAxis3`
: This field is not used.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventFixedPtDeltaAxis1`
: Key to access a field that contains scrolling data. The scrolling data represents a line-based or pixel-based change in vertical position since the last scrolling event from a Mighty Mouse scroller or a single-wheel mouse scroller. The scrolling data uses a fixed-point 16.16 signed integer format. For example, if the field contains a value of 1.0, the integer 0x00010000 is returned by `CGEventGetIntegerValueField`. If this key is passed to `CGEventGetDoubleValueField`, the fixed-point value is converted to a double value.

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventFixedPtDeltaAxis2`
: Key to access a field that contains scrolling data. The scrolling data represents a line-based or pixel-based change in horizontal position since the last scrolling event from a Mighty Mouse scroller. The scrolling data uses a fixed-point 16.16 signed integer format. For example, if the field contains a value of 1.0, the integer 0x00010000 is returned by `CGEventGetIntegerValueField`. If this key is passed to `CGEventGetDoubleValueField`, the fixed-point value is converted to a double value.

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventFixedPtDeltaAxis3`
: This field is not used.

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventPointDeltaAxis1`
Key to access an integer field that contains pixel-based scrolling data. The scrolling data represents the change in vertical position since the last scrolling event from a Mighty Mouse scroller or a single-wheel mouse scroller.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollWheelEventPointDeltaAxis2`
Key to access an integer field that contains pixel-based scrolling data. The scrolling data represents the change in horizontal position since the last scrolling event from a Mighty Mouse scroller.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollWheelEventPointDeltaAxis3`
This field is not used.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollWheelEventInstantMouser`
Key to access an integer field that indicates whether the event should be ignored by the Inkwell subsystem. If the value is non-zero, the event should be ignored.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointX`
Key to access an integer field that contains the absolute X coordinate in tablet space at full tablet resolution.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointY`
Key to access an integer field that contains the absolute Y coordinate in tablet space at full tablet resolution.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointZ`
Key to access an integer field that contains the absolute Z coordinate in tablet space at full tablet resolution.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointButtons`
Key to access an integer field that contains the tablet button state. Bit 0 is the first button, and a set bit represents a closed or pressed button. Up to 16 buttons are supported.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointPressure`
Key to access a double field that contains the tablet pen pressure. A value of 0.0 represents no pressure, and 1.0 represents maximum pressure.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventTiltX`
Key to access a double field that contains the horizontal tablet pen tilt. A value of 0.0 represents no tilt, and 1.0 represents maximum tilt.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventTiltY`
Key to access a double field that contains the vertical tablet pen tilt. A value of 0.0 represents no tilt, and 1.0 represents maximum tilt.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventRotation`
Key to access a double field that contains the tablet pen rotation.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventTangentialPressure`
Key to access a double field that contains the tangential pressure on the device. A value of 0.0 represents no pressure, and 1.0 represents maximum pressure.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventDeviceID`
Key to access an integer field that contains the system-assigned unique device ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventVendor1`
Key to access an integer field that contains a vendor-specified value.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventVendor2`
Key to access an integer field that contains a vendor-specified value.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventVendor3`
Key to access an integer field that contains a vendor-specified value.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorID`
Key to access an integer field that contains the vendor-defined ID, typically the USB vendor ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventTabletID`
Key to access an integer field that contains the vendor-defined tablet ID, typically the USB product ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventPointerID`
Key to access an integer field that contains the vendor-defined ID of the pointing device.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventDeviceID`
Key to access an integer field that contains the system-assigned device ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventSystemTabletID`
Key to access an integer field that contains the system-assigned unique tablet ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorPointerType`
Key to access an integer field that contains the vendor-assigned pointer type.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorPointerSerialNumber`
Key to access an integer field that contains the vendor-defined pointer serial number.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorUniqueID`
Key to access an integer field that contains the vendor-defined unique ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventCapabilityMask`
Key to access an integer field that contains the device capabilities mask.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventPointerType`
Key to access an integer field that contains the pointer type.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletProximityEventEnterProximity`
Key to access an integer field that indicates whether the pen is in proximity to the tablet. The value is non-zero if the pen is in proximity to the tablet and zero when leaving the tablet.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventTargetProcessSerialNumber`
Key to access a field that contains the event target process serial number. The value is a 64-bit long word.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventTargetUnixProcessID`
Key to access a field that contains the event target Unix process ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventSourceUnixProcessID`
Key to access a field that contains the event source Unix process ID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventSourceUserData`
Key to access a field that contains the event source user-supplied data, up to 64 bits.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventSourceUserID`
Key to access a field that contains the event source Unix effective UID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventSourceGroupID`
Key to access a field that contains the event source Unix effective GID.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventSourceStateID`
Key to access a field that contains the event source state ID used to create this event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGScrollWheelEventIsContinuous`
Key to access an integer field that indicates whether a scrolling event contains continuous, pixel-based scrolling data. The value is non-zero when the scrolling data is pixel-based and zero when the scrolling data is line-based.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

Discussion
These constants are used as keys to access certain specialized event fields when using low-level accessor functions such as `CGEventGetIntegerValueField` (page 1577), `CGEventSetIntegerValueField` (page 1584), `CGEventGetDoubleValueField` (page 1576), and `CGEventSetDoubleValueField` (page 1583).

Event Filter Masks

Specify masks for classes of low-level events that can be filtered during event suppression states.

```
enum CGEventFilterMask {
    kCGEventFilterMaskPermitLocalMouseEvents = 0x00000001,
    kCGEventFilterMaskPermitLocalKeyboardEvents = 0x00000002,
    kCGEventFilterMaskPermitSystemDefinedEvents = 0x00000004,
    kCGEventFilterMaskPermitAllEvents = kCGEventFilterMaskPermitLocalMouseEvents
 | kCGEventFilterMaskPermitLocalKeyboardEvents |
kCGEventFilterMaskPermitSystemDefinedEvents
};
typedef uint32_t CGEventFilterMask;
```

Event Flags

Constants that indicate the modifier key state at the time an event is created, as well as other event-related states.

```
enum _CGEventFlags {
    kCGEventFlagMaskAlphaShift =    NX_ALPHASHIFTMASK,
    kCGEventFlagMaskShift =         NX_SHIFTMASK,
    kCGEventFlagMaskControl =       NX_CONTROLMASK,
    kCGEventFlagMaskAlternate =     NX_ALTERNATEMASK,
    kCGEventFlagMaskCommand =       NX_COMMANDMASK,
    kCGEventFlagMaskHelp =          NX_HELPMASK,
    kCGEventFlagMaskSecondaryFn =   NX_SECONDARYFNMASK,
    kCGEventFlagMaskNumericPad =    NX_NUMERICPADMASK,
    kCGEventFlagMaskNonCoalesced =  NX_NONCOALSESCEDMASK
};
typedef uint64_t CGEventFlags;
```

Constants

`kCGEventFlagMaskAlphaShift`
Indicates that the Caps Lock key is down for a keyboard, mouse, or flag-changed event.
Available in Mac OS X v10.4 and later.
Declared in `CGEventTypes.h`.

`kCGEventFlagMaskShift`
Indicates that the Shift key is down for a keyboard, mouse, or flag-changed event.
Available in Mac OS X v10.4 and later.
Declared in `CGEventTypes.h`.

`kCGEventFlagMaskControl`
Indicates that the Control key is down for a keyboard, mouse, or flag-changed event.
Available in Mac OS X v10.4 and later.
Declared in `CGEventTypes.h`.

`kCGEventFlagMaskAlternate`
Indicates that the Alt or Option key is down for a keyboard, mouse, or flag-changed event.
Available in Mac OS X v10.4 and later.
Declared in `CGEventTypes.h`.

`kCGEventFlagMaskCommand`
Indicates that the Command key is down for a keyboard, mouse, or flag-changed event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskHelp`
Indicates that the Help modifier key is down for a keyboard, mouse, or flag-changed event. This key is not present on most keyboards, and is different than the Help key found in the same row as Home and Page Up.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskSecondaryFn`
Indicates that the Fn (Function) key is down for a keyboard, mouse, or flag-changed event. This key is found primarily on laptop keyboards.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskNumericPad`
Identifies key events from the numeric keypad area on extended keyboards.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskNonCoalesced`
Indicates that mouse and pen movement events are not being coalesced.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion
These constants specify masks for the bits in an event flags bit mask. Event flags indicate the modifier key state at the time an event is created, as well as other event-related states. Event flags are used in accessor functions such as `CGEventGetFlags` (page 1577), `CGEventSetFlags` (page 1583), and `CGEventSourceFlagsState` (page 1588).

Event Source States

Constants that specify the possible source states of an event source.

```
enum {
    kCGEventSourceStatePrivate = -1,
    kCGEventSourceStateCombinedSessionState = 0,
    kCGEventSourceStateHIDSystemState = 1
};
typedef uint32_t CGEventSourceStateID;
```

Constants
`kCGEventSourceStatePrivate`
Specifies that an event source should use a private event state table.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventSourceStateCombinedSessionState`
: Specifies that an event source should use the event state table that reflects the combined state of all event sources posting to the current user login session.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceStateHIDSystemState`
: Specifies that an event source should use the event state table that reflects the combined state of all hardware event sources posting from the HID system.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion
A source state refers to a global event state table. These tables contain accumulated information on modifier flag state, keyboard key state, mouse button state, and related internal parameters placed in effect by posting events with associated sources.

Two pre-existing event state tables are defined:

- The `kCGEventSourceStateCombinedSessionState` table reflects the combined state of all event sources posting to the current user login session. If your program is posting events from within a login session, you should use this source state when you create an event source.
- The `kCGEventSourceStateHIDSystemState` table reflects the combined state of all hardware event sources posting from the HID system. If your program is a daemon or a user space device driver interpreting hardware state and generating events, you should use this source state when you create an event source.

Specialized applications such as remote control programs may want to generate and track event source state independent of other processes. These programs should use the `kCGEventSourceStatePrivate` value in creating their event source. An independent state table and unique source state ID (`CGEventSourceStateID`) are created to track the event source's state. This independent state table is owned by the creating event source and released with it.

Event Source Token

Specifies any input event type.

```
#define kCGAnyInputEventType ((CGEventType)(~0))
```

Constants
`kCGAnyInputEventType`
: A constant that specifies any input event type.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion
This constant is typically used with the function `CGEventSourceSecondsSinceLastEventType` (page 1592) to specify that you want the elapsed time since the last input event of any type.

Event Suppression States

Specify the event suppression states that can occur after posting an event.

```
enum CGEventSuppressionState {
    kCGEventSuppressionStateSuppressionInterval = 0,
    kCGEventSuppressionStateRemoteMouseDrag = 1,
    kCGNumberOfEventSuppressionStates = 2
};
typedef uint32_t CGEventSuppressionState;
```

Constants

`kCGEventSuppressionStateSuppressionInterval`

Specifies that certain local hardware events may be suppressed for a short interval after posting an event.

Available in Mac OS X v10.3 and later.

Declared in `CGRemoteOperation.h`.

`kCGEventSuppressionStateRemoteMouseDrag`

Specifies that certain local hardware events may be suppressed during a mouse drag operation (mouse movement with the left or only mouse button down).

Available in Mac OS X v10.3 and later.

Declared in `CGRemoteOperation.h`.

Discussion

These constants specify the types of event suppression intervals during which an event filter is applied after posting an event.

Event Tap Locations

Constants that specify possible tapping points for events.

```
enum _CGEventTapLocation {
    kCGHIDEventTap = 0,
    kCGSessionEventTap,
    kCGAnnotatedSessionEventTap
};
typedef uint32_t CGEventTapLocation;
```

Constants

`kCGHIDEventTap`

Specifies that an event tap is placed at the point where HID system events enter the window server.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGSessionEventTap`

Specifies that an event tap is placed at the point where HID system and remote control events enter a login session.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGAnnotatedSessionEventTap`
: Specifies that an event tap is placed at the point where session events have been annotated to flow to an application.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion
In addition to the three tapping points described above, an event tap may also be placed where annotated events are delivered to a specific application. For more information, see the function `CGEventTapCreateForPSN` (page 1597).

Event Tap Options

Constants that specify whether a new event tap is an active filter or a passive listener.

```
enum _CGEventTapOptions {
    kCGEventTapOptionDefault = 0x00000000,
    kCGEventTapOptionListenOnly = 0x00000001
};
typedef uint32_t CGEventTapOptions;
```

Constants
`kCGEventTapOptionDefault`
: Specifies that a new event tap is an active filter. (Applications targeting Mac OS X v10.4 should use the literal value to create an active filter event tap, as this constant was omitted from the header.)

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

`kCGEventTapOptionListenOnly`
: Specifies that a new event tap is a passive listener.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion
When you create an event tap, you indicate whether it is a passive listener or active event filter. A passive listener receives events but cannot modify or divert them. An active filter may pass an event through unmodified, modify an event, or discard an event.

Event Tap Placement

Constants that specify where a new event tap is inserted into the list of active event taps.

```
enum _CGEventTapPlacement {
    kCGHeadInsertEventTap = 0,
    kCGTailAppendEventTap
};
typedef uint32_t CGEventTapPlacement;
```

Constants

`kCGHeadInsertEventTap`
Specifies that a new event tap should be inserted before any pre-existing event taps at the same location.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTailAppendEventTap`
Specifies that a new event tap should be inserted after any pre-existing event taps at the same location.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion
Event taps may be inserted at a specified location at the head of pre-existing filters, or appended after any pre-existing filters.

Event Types

Constants that specify the different types of input events.

```
enum _CGEventType {
    kCGEventNull                = NX_NULLEVENT,
    kCGEventLeftMouseDown       = NX_LMOUSEDOWN,
    kCGEventLeftMouseUp         = NX_LMOUSEUP,
    kCGEventRightMouseDown      = NX_RMOUSEDOWN,
    kCGEventRightMouseUp        = NX_RMOUSEUP,
    kCGEventMouseMoved          = NX_MOUSEMOVED,
    kCGEventLeftMouseDragged    = NX_LMOUSEDRAGGED,
    kCGEventRightMouseDragged   = NX_RMOUSEDRAGGED,
    kCGEventKeyDown             = NX_KEYDOWN,
    kCGEventKeyUp               = NX_KEYUP,
    kCGEventFlagsChanged        = NX_FLAGSCHANGED,
    kCGEventScrollWheel         = NX_SCROLLWHEELMOVED,
    kCGEventTabletPointer       = NX_TABLETPOINTER,
    kCGEventTabletProximity     = NX_TABLETPROXIMITY,
    kCGEventOtherMouseDown      = NX_OMOUSEDOWN,
    kCGEventOtherMouseUp        = NX_OMOUSEUP,
    kCGEventOtherMouseDragged   = NX_OMOUSEDRAGGED,
    kCGEventTapDisabledByTimeout = 0xFFFFFFFE,
    kCGEventTapDisabledByUserInput = 0xFFFFFFFF
};
typedef uint32_t CGEventType;
```

Constants

`kCGEventNull`
Specifies a null event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventLeftMouseDown`
Specifies a mouse down event with the left button.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventLeftMouseUp`
Specifies a mouse up event with the left button.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventRightMouseDown`
Specifies a mouse down event with the right button.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventRightMouseUp`
Specifies a mouse up event with the right button.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventMouseMoved`
Specifies a mouse moved event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventLeftMouseDragged`
Specifies a mouse drag event with the left button down.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventRightMouseDragged`
Specifies a mouse drag event with the right button down.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventKeyDown`
Specifies a key down event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventKeyUp`
Specifies a key up event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagsChanged`
Specifies a key changed event for a modifier or status key.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventScrollWheel`
Specifies a scroll wheel moved event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventTabletPointer`
Specifies a tablet pointer event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventTabletProximity`
Specifies a tablet proximity event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventOtherMouseDown`
Specifies a mouse down event with one of buttons 2-31.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventOtherMouseUp`
Specifies a mouse up event with one of buttons 2-31.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventOtherMouseDragged`
Specifies a mouse drag event with one of buttons 2-31 down.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventTapDisabledByTimeout`
Specifies an event indicating the event tap is disabled because of timeout.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventTapDisabledByUserInput`
Specifies an event indicating the event tap is disabled because of user input.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion
These constants are used:

- In the functions `CGEventTapCreate` (page 1595) and `CGEventTapCreateForPSN` (page 1597) to specify the events of interest for the new event tap.
- To indicate the event type passed to your event tap callback function.
- In the function `CGEventCreateMouseEvent` (page 1574) to specify the type of mouse event.
- In the functions `CGEventGetType` (page 1579) and `CGEventSetType` (page 1586) to identify the event type.
- In the functions `CGEventSourceCounterForEventType` (page 1587) and `CGEventSourceSecondsSinceLastEventType` (page 1592) to indicate the event type.

Note that tablet devices may generate mouse events with embedded tablet data, or tablet pointer and proximity events. Tablet mouse events allow tablets to be used with applications that are not tablet-aware.

Event Type Mask

Specifies an event mask that represents all event types.

```
#define kCGEventMaskForAllEvents (~(CGEventMask)0)
```

Constants

`kCGEventMaskForAllEvents`

An event mask that specifies all event types.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

This constant is typically used with the functions `CGEventTapCreate` (page 1595) and `CGEventTapCreateForPSN` (page 1597) to register an event tap that observes all input events.

Mouse Buttons

Constants that specify buttons on a one, two, or three-button mouse.

```
enum _CGMouseButton {
    kCGMouseButtonLeft = 0,
    kCGMouseButtonRight = 1,
    kCGMouseButtonCenter = 2
};
typedef uint32_t CGMouseButton;
```

Constants

`kCGMouseButtonLeft`

Specifies the only mouse button on a one-button mouse, or the left mouse button on a two-button or three-button mouse.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseButtonRight`

Specifies the right mouse button on a two-button or three-button mouse.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseButtonCenter`

Specifies the center mouse button on a three-button mouse.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

Quartz supports up to 32 mouse buttons. The first three buttons are specified using these three constants. Additional buttons are specified in USB order using the integers 3 to 31.

These constants are used:

- In the function `CGEventCreateMouseEvent` (page 1574) to specify the button that's changing state.
- In the function `CGEventSourceButtonState` (page 1586) to specify the button that's being tested.
- To specify the value of the `kCGMouseEventButtonNumber` event field when modifying an event.

Mouse Subtypes

Constants used with the `kCGMouseEventSubtype` event field.

```
enum _CGEventMouseSubtype {
    kCGEventMouseSubtypeDefault = 0,
    kCGEventMouseSubtypeTabletPoint = 1,
    kCGEventMouseSubtypeTabletProximity = 2
};
typedef uint32_t CGEventMouseSubtype;
```

Constants

`kCGEventMouseSubtypeDefault`
Specifies that the event is an ordinary mouse event, and does not contain additional tablet device information.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventMouseSubtypeTabletPoint`
Specifies that the mouse event originated from a tablet device, and that the various `kCGTabletEvent` field selectors may be used to obtain tablet-specific data from the mouse event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventMouseSubtypeTabletProximity`
Specifies that the mouse event originated from a tablet device with the pen in proximity but not necessarily touching the tablet, and that the various `kCGTabletProximity` field selectors may be used to obtain tablet-specific data from the mouse event. This is often used with mouse move events originating from a tablet.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

Tablets may generate specially annotated mouse events that contain values associated with the `kCGMouseEventSubtype` event field. To learn how to set these values, see the function `CGEventSetIntegerValueField` (page 1584).

Scrolling Event Units

Constants that specify the unit of measurement for a scrolling event.

```
enum {
    kCGScrollEventUnitPixel = 0,
    kCGScrollEventUnitLine = 1
};
typedef uint32_t CGScrollEventUnit;
```

Constants

`kCGScrollEventUnitPixel`

Specifies that the unit of measurement is pixels.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollEventUnitLine`

Specifies that the unit of measurement is lines.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

Discussion

You may pass one of these constants to the function `CGEventCreateScrollWheelEvent` (page 1575) to specify the unit of measurement for the event. The constant `kCGScrollEventUnitPixel` produces an event that most applications interpret as a smooth scrolling event. By default, the scale is about ten pixels per line. You can alter the scale with the function `CGEventSourceSetPixelsPerLine` (page 1594).

Speech Synthesis Manager Reference

Framework: ApplicationServices/ApplicationServices.h
Declared in SpeechSynthesis.h

Overview

The Speech Synthesis Manager, formerly called the Speech Manager, is the part of the Mac OS that provides a standardized method for Macintosh applications to generate synthesized speech. For example, you may want your application to incorporate the capability to speak its dialog box messages to the user. A word-processing application might use the Speech Synthesis Manager to implement a command that speaks a selected section of a document to the user. Because sound samples can take up large amounts of room on disk, using text in place of sampled sound is extremely efficient, and so a multimedia application might use the Speech Synthesis Manager to provide a narration of a QuickTime movie instead of including sampled-sound data on a movie track.

Mac OS X v10.5 introduces native support for performing speech synthesis tasks using Core Foundation-based objects, such as speaking text represented as `CFString` objects and managing speech channel properties using a `CFDictionary`-based property dictionary. You should begin using the new, Core Foundation-based programming interfaces as soon as it's convenient, because future synthesizers will accept Core Foundation strings and data structures directly through the speech synthesis framework. In the meantime, existing buffer-based clients and synthesizers will continue to work as before, with strings and other data structures getting automatically converted as necessary.

Functions by Task

Changing Speech Attributes

`SetSpeechInfo` (page 1650)
Changes a setting of a particular speech channel.

`SetSpeechProperty` (page 1651)
Sets the value of the specified speech-channel property.

`SetSpeechPitch` (page 1651)
Sets the speech pitch on a designated speech channel.

`SetSpeechRate` (page 1652)
Sets the speech rate of a designated speech channel.

Converting Text To Phonemes

`TextToPhonemes` (page 1659)
Converts a buffer of textual data into phonemic data.

`CopyPhonemesFromText` (page 1633) Deprecated in Mac OS X v10.4
Converts the specified text string into its equivalent phonemic representation.

Installing a Pronunciation Dictionary

`UseDictionary` (page 1660)
Installs the designated dictionary into a speech channel.

`UseSpeechDictionary` (page 1661)
Registers a speech dictionary with a speech channel.

Managing Speech Channels

`DisposeSpeechChannel` (page 1635)
Disposes of an existing speech channel.

`NewSpeechChannel` (page 1646)
Creates a new speech channel.

Obtaining Information About Speech and Speech Channels

`CopySpeechProperty` (page 1634)
Gets the value associated with the specified property of a speech channel.

`GetSpeechInfo` (page 1638)
Gets information about a designated speech channel.

`GetSpeechPitch` (page 1639)
Gets a speech channel's current speech pitch.

`GetSpeechRate` (page 1640)
Gets a speech channel's current speech rate.

`SpeechBusy` (page 1656)
Determines whether any channels of speech are currently synthesizing speech.

`SpeechBusySystemWide` (page 1656)
Determines if any speech is currently being synthesized in your application or elsewhere on the computer.

`SpeechManagerVersion` (page 1657)
Determines the current version of the Speech Synthesis Manager installed in the system.

Getting Information About Voices

`CountVoices` (page 1634)
Determines how many voices are available.

`GetIndVoice` (page 1638)
Gets a voice specification structure for a voice by passing an index to the `GetIndVoice` function.

`GetVoiceDescription` (page 1640)
Gets a description of a voice by using the `GetVoiceDescription` function.

`GetVoiceInfo` (page 1641)
Gets the same information about a voice that the `GetVoiceDescription` function provides, or to determine in which file and resource a voice is stored.

`MakeVoiceSpec` (page 1645)
Sets the fields of a voice specification structure.

Starting, Stopping, and Pausing Speech

`ContinueSpeech` (page 1632)
Resumes speech paused by the `PauseSpeechAt` function.

`PauseSpeechAt` (page 1649)
Pauses speech on a speech channel.

`SpeakBuffer` (page 1652)
Speaks a buffer of text, using certain flags to control speech behavior.

`SpeakString` (page 1654)
Begins speaking a text string.

`SpeakText` (page 1655)
Begins speaking a buffer of text.

`StopSpeech` (page 1657)
Terminates speech immediately on the specified channel.

`StopSpeechAt` (page 1658)
Terminates speech delivery on a specified channel either immediately or at the end of the current word or sentence.

`SpeakCFString` (page 1653) Deprecated in Mac OS X v10.5
Begins speaking a string represented as a `CFString` object.

Creating, Invoking, and Disposing Universal Procedure Pointers

`DisposeSpeechDoneUPP` (page 1635)
Disposes of a universal procedure pointer (UPP) to a speech-done callback function.

`DisposeSpeechErrorUPP` (page 1636)
Disposes of a universal procedure pointer (UPP) to an error callback function.

`DisposeSpeechPhonemeUPP` (page 1636)
Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

`DisposeSpeechSyncUPP` (page 1637)
Disposes of a universal procedure pointer (UPP) to a synchronization callback function.

`DisposeSpeechTextDoneUPP` (page 1637)
Disposes of a universal procedure pointer (UPP) to a text-done callback function.

`DisposeSpeechWordUPP` (page 1637)
Disposes of a universal procedure pointer (UPP) to a word callback function.

`InvokeSpeechDoneUPP` (page 1642)
Invokes your speech-done callback function.

`InvokeSpeechErrorUPP` (page 1643)
Invokes your error callback function.

`InvokeSpeechPhonemeUPP` (page 1643)
Invokes your phoneme callback function.

`InvokeSpeechSyncUPP` (page 1644)
Invokes your synchronization callback function.

`InvokeSpeechTextDoneUPP` (page 1644)
Invokes your text-done callback function.

`InvokeSpeechWordUPP` (page 1645)
Invokes your word callback function.

`NewSpeechDoneUPP` (page 1646)
Creates a new universal procedure pointer (UPP) to a speech-done callback function.

`NewSpeechErrorUPP` (page 1647)
Creates a new universal procedure pointer to an error callback function.

`NewSpeechPhonemeUPP` (page 1647)
Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

`NewSpeechSyncUPP` (page 1648)
Creates a new universal procedure pointer (UPP) to a synchronization callback function.

`NewSpeechTextDoneUPP` (page 1648)
Creates a new universal procedure pointer (UPP) to a text-done callback function.

`NewSpeechWordUPP` (page 1649)
Creates a new universal procedure pointer (UPP) to a word callback function.

Functions

ContinueSpeech

Resumes speech paused by the `PauseSpeechAt` function.

```
OSErr ContinueSpeech (
   SpeechChannel chan
);
```

Parameters

chan
The paused speech channel on which speech is to be resumed.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
At any time after the `PauseSpeechAt` function is called, the `ContinueSpeech` function can be called to continue speaking from the beginning of the word in which speech paused. Calling `ContinueSpeech` on a channel that is not currently in a paused state has no effect on the speech channel or on future calls to the `PauseSpeechAt` function. If you call `ContinueSpeech` on a channel before a pause is effective, `ContinueSpeech` cancels the pause.

If the `PauseSpeechAt` function stopped speech in the middle of a word, the Speech Synthesis Manager will start speaking that word from the beginning when you call `ContinueSpeech`.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

CopyPhonemesFromText

Converts the specified text string into its equivalent phonemic representation.

```
OSErr CopyPhonemesFromText (
   SpeechChannel chan,
   CFStringRef text,
   CFStringRef * phonemes
);
```

Parameters

chan
A speech channel whose associated synthesizer and properties are to be used in the conversion process.

text
The text from which to extract phonemic data.

phonemes
On return, a `CFString` object that contains the extracted phonemic data. The caller is responsible for releasing this object.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `CopyPhonemesFromText` function is the Core Foundation-based equivalent of the `TextToPhonemes` (page 1659) function.

Converting textual data into phonemic data is particularly useful during application development, when you might wish to adjust phrases that your application generates to produce smoother speech. By first converting the target phrase into phonemes, you can see what the synthesizer will try to speak. Then you need correct only the parts that would not have been spoken the way you want.

The data the `CopyPhonemesFromText` function stores in the `phonemes` parameter corresponds precisely to the phonemes that would be spoken had the input text been sent to `SpeakCFString` instead. All current property settings for the speech channel specified by `chan` are applied to the converted speech. No callbacks are generated while the `CopyPhonemesFromText` function is generating its output.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

CopySpeechProperty

Gets the value associated with the specified property of a speech channel.

```
OSErr CopySpeechProperty (
   SpeechChannel chan,
   CFStringRef property,
   CFTypeRef * object
);
```

Parameters

chan

The speech channel with which the specified property is associated.

property

A speech-channel property about which information is being requested. See "`Speech-Channel Properties`" (page 1692) for information on the properties you can specify.

object

On return, a pointer to a Core Foundation object that holds the value of the specified property. The type of the object depends on the specific property passed in. For some properties, the value of *object* can be `NULL`. When the returned object is a `CFDictionary` object, you can use `CFDictionary` functions, such as `CFDictionaryGetValue`, to retrieve the values associated with the keys that are associated with the specified property.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `CopySpeechProperty` function is the Core Foundation-based equivalent of the `GetSpeechInfo` (page 1638) function.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

CountVoices

Determines how many voices are available.

```
OSErr CountVoices (
   SInt16 *numVoices
);
```

Parameters

numVoices

On exit, a pointer to the number of voices that the application can use.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `CountVoices` function returns, in the `numVoices` parameter, the number of voices available. The application can then use this information to call the `GetIndVoice` function to obtain voice specification structures for one or more of the voices.

Each time `CountVoices` is called, the Speech Synthesis Manager searches for new voices.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

DisposeSpeechChannel

Disposes of an existing speech channel.

```
OSErr DisposeSpeechChannel (
   SpeechChannel chan
);
```

Parameters

chan
The speech channel to dispose of.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `DisposeSpeechChannel` function disposes of the speech channel specified in the `chan` parameter and releases all memory the channel occupies. If the speech channel specified is producing speech, then the `DisposeSpeechChannel` function immediately stops speech before disposing of the channel. If you have defined a text-done callback function or a speech-done callback function, the function will not be called before the channel is disposed of.

The Speech Synthesis Manager releases any speech channels that have not been explicitly disposed of by an application when the application quits. In general, however, your application should dispose of any speech channels it has created whenever it receives a suspend event. This ensures that other applications can take full advantage of Speech Synthesis Manager and Sound Manager capabilities.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

DisposeSpeechDoneUPP

Disposes of a universal procedure pointer (UPP) to a speech-done callback function.

```
void DisposeSpeechDoneUPP (
   SpeechDoneUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechErrorUPP

Disposes of a universal procedure pointer (UPP) to an error callback function.

```
void DisposeSpeechErrorUPP (
   SpeechErrorUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechPhonemeUPP

Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

```
void DisposeSpeechPhonemeUPP (
   SpeechPhonemeUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

DisposeSpeechSyncUPP

Disposes of a universal procedure pointer (UPP) to a synchronization callback function.

```
void DisposeSpeechSyncUPP (
   SpeechSyncUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

SpeechSynthesis.h

DisposeSpeechTextDoneUPP

Disposes of a universal procedure pointer (UPP) to a text-done callback function.

```
void DisposeSpeechTextDoneUPP (
   SpeechTextDoneUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

SpeechSynthesis.h

DisposeSpeechWordUPP

Disposes of a universal procedure pointer (UPP) to a word callback function.

```
void DisposeSpeechWordUPP (
   SpeechWordUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In
SpeechSynthesis.h

GetIndVoice

Gets a voice specification structure for a voice by passing an index to the GetIndVoice function.

```
OSErr GetIndVoice (
   SInt16 index,
   VoiceSpec *voice
);
```

Parameters

index

The index of the voice for which to obtain a voice specification structure. This number must range from 1 to the total number of voices, as returned by the CountVoices function.

voice

A pointer to the voice specification structure whose fields are to be filled in.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The GetIndVoice function returns, in the voice specification structure pointed to by the voice parameter, a specification of the voice whose index is provided in the index parameter. Your application should make no assumptions about the order in which voices are indexed.

Your application should not add, remove, or modify a voice and then call the GetIndVoice function with an index value other than 1. To allow the Speech Synthesis Manager to update its information about voices, your application should always either call the CountVoices function or call the GetIndVoice function with an index value of 1 after adding, removing, or modifying a voice or after a time at which the user might have done so.

If you specify an index value beyond the number of available voices, the GetIndVoice function returns a voiceNotFound error.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechSynthesis.h

GetSpeechInfo

Gets information about a designated speech channel.

```
OSErr GetSpeechInfo (
   SpeechChannel chan,
   OSType selector,
   void *speechInfo
);
```

Parameters

chan

The speech channel about which information is being requested.

selector

A speech information selector that indicates the type of information being requested.

For a complete list of speech information selectors, see "Speech-Channel Information Constants" (page 1685). This list indicates how your application should set the `speechInfo` parameter for each selector.

speechInfo

A pointer whose meaning depends on the speech information selector specified in the `selector` parameter.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion

The `GetSpeechInfo` function returns, in the data structure pointed to by the `speechInfo` parameter, the type of information requested by the `selector` parameter as it applies to the speech channel specified in the `chan` parameter.

The format of the data structure specified by the `speechInfo` parameter depends on the selector you choose. For example, a selector might require that your application allocate a block of memory of a certain size and pass a pointer to that block. Another selector might require that `speechInfo` be set to the address of a handle variable. In this case, the `GetSpeechInfo` function would allocate a relocatable block of memory and change the handle variable specified to reference the block.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

GetSpeechPitch

Gets a speech channel's current speech pitch.

```
OSErr GetSpeechPitch (
   SpeechChannel chan,
   Fixed *pitch
);
```

Parameters

chan

The speech channel whose pitch you wish to determine.

pitch

On return, a pointer to the current pitch of the voice in the speech channel, expressed as a fixed-point frequency value.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
Typical voice frequencies range from around 90 hertz for a low-pitched male voice to perhaps 300 hertz for a high-pitched child's voice. These frequencies correspond to approximate pitch values in the ranges of 30.000 to 40.000 and 55.000 to 65.000, respectively.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

GetSpeechRate

Gets a speech channel's current speech rate.

```
OSErr GetSpeechRate (
   SpeechChannel chan,
   Fixed *rate
);
```

Parameters

chan
The speech channel whose rate you wish to determine.

rate
On return, a pointer to the speech channel's speech rate in words per minute, expressed as an integer value.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

GetVoiceDescription

Gets a description of a voice by using the `GetVoiceDescription` function.

```
OSErr GetVoiceDescription (
   const VoiceSpec *voice,
   VoiceDescription *info,
   long infoLength
);
```

Parameters

voice

A pointer to the voice specification structure identifying the voice to be described, or `NULL` to obtain a description of the system default voice.

info

A pointer to a voice description structure. If this parameter is `NULL`, the function does not fill in the fields of the voice description structure; instead, it simply determines whether the `voice` parameter specifies an available voice and, if not, returns a `voiceNotFound` error.

infoLength

The length, in bytes, of the voice description structure. In the current version of the Speech Synthesis Manager, the voice description structure contains 362 bytes. However, you should always use the `SizeOf` function to determine the length of this structure.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion

The `GetVoiceDescription` function fills out the voice description structure pointed to by the `info` parameter with the correct information for the voice specified by the `voice` parameter. It fills in the `length` field of the voice description structure with the number of bytes actually copied. This value will always be less than or equal to the value that your application passes in `infoLength` before calling `GetVoiceDescription`. This scheme allows applications targeted for the current version of the Speech Synthesis Manager to work on future versions that might have longer voice description structures; it also allows you to write code for future versions of the Speech Synthesis Manager that will also run on computers that support only the current version.

If the voice specification structure does not identify an available voice, `GetVoiceDescription` returns a `voiceNotFound` error.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

GetVoiceInfo

Gets the same information about a voice that the `GetVoiceDescription` function provides, or to determine in which file and resource a voice is stored.

```
OSErr GetVoiceInfo (
   const VoiceSpec *voice,
   OSType selector,
   void *voiceInfo
);
```

Parameters

voice

A pointer to the voice specification structure identifying the voice about which your application requires information, or `NULL` to obtain information on the system default voice.

selector

A specification of the type of data being requested. For current versions of the Speech Synthesis Manager, you should set this field either to `soVoiceDescription`, if you would like to use the `GetVoiceInfo` function to mimic the `GetVoiceDescription` function, or to `soVoiceFile`, if you would like to obtain information about the location of a voice on disk.

voiceInfo

A pointer to the appropriate data structure. If the selector is `soVoiceDescription`, then `voiceInfo` should be a pointer to a voice description structure, and the `length` field of the structure should be set to the length of the voice description structure. If the selector is `soVoiceFile`, then `voiceInfo` should be a pointer to a voice file information structure.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
This function is intended primarily for use by synthesizers, but an application can call it too.

The `GetVoiceInfo` function accepts a selector in the `selector` parameter that determines the type of information you wish to obtain about the voice specified in the `voice` parameter. The function then fills the fields of the data structure appropriate to the selector you specify in the `voiceInfo` parameter.

If the voice specification is invalid, `GetVoiceInfo` returns a `voiceNotFound` error. If there is not enough memory to load the voice into memory to obtain information about it, `GetVoiceInfo` returns the result code `memFullErr`.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechDoneUPP

Invokes your speech-done callback function.

```
void InvokeSpeechDoneUPP (
   SpeechChannel chan,
   SRefCon refCon,
   SpeechDoneUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechDoneUPP` function, because the system calls your speech-done callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechErrorUPP

Invokes your error callback function.

```
void InvokeSpeechErrorUPP (
   SpeechChannel chan,
   SRefCon refCon,
   OSErr theError,
   long bytePos,
   SpeechErrorUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechErrorUPP` function, because the system calls your error callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechPhonemeUPP

Invokes your phoneme callback function.

```
void InvokeSpeechPhonemeUPP (
   SpeechChannel chan,
   SRefCon refCon,
   SInt16 phonemeOpcode,
   SpeechPhonemeUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechPhonemeUPP` function, because the system calls your phoneme callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechSyncUPP

Invokes your synchronization callback function.

```
void InvokeSpeechSyncUPP (
   SpeechChannel chan,
   SRefCon refCon,
   OSType syncMessage,
   SpeechSyncUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechSyncUPP` function, because the system calls your synchronization callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechTextDoneUPP

Invokes your text-done callback function.

```
void InvokeSpeechTextDoneUPP (
   SpeechChannel chan,
   SRefCon refCon,
   const void **nextBuf,
   unsigned long *byteLen,
   SInt32 *controlFlags,
   SpeechTextDoneUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechTextDoneUPP` function, because the system calls your text-done callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

InvokeSpeechWordUPP

Invokes your word callback function.

```
void InvokeSpeechWordUPP (
   SpeechChannel chan,
   SRefCon refCon,
   unsigned long wordPos,
   UInt16 wordLen,
   SpeechWordUPP userUPP
);
```

Discussion
You should not need to call the `InvokeSpeechWordUPP` function, because the system calls your word callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

MakeVoiceSpec

Sets the fields of a voice specification structure.

```
OSErr MakeVoiceSpec (
   OSType creator,
   OSType id,
   VoiceSpec *voice
);
```

Parameters

creator
The ID of the synthesizer that your application requires.

id
The ID of the voice on the synthesizer specified by the `creator` parameter.

voice
A pointer to the voice specification structure whose fields are to be filled in.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
A voice specification structure is a unique voice ID used by the Speech Synthesis Manager. Most voice management functions expect to be passed a pointer to a voice specification structure. When you already know the creator and ID for a voice, you should use the `MakeVoiceSpec` function to create such a structure rather than filling in the fields of one directly. On exit, the voice specification structure pointed to by the `voice` parameter contains the appropriate values. You should never set the fields of such a structure directly.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechSynthesis.h

NewSpeechChannel

Creates a new speech channel.

```
OSErr NewSpeechChannel (
   VoiceSpec *voice,
   SpeechChannel *chan
);
```

Parameters

voice

A pointer to the voice specification structure corresponding to the voice to be used for the new speech channel. Pass NULL to create a speech channel using the system default voice. Specifying a voice means the initial speaking rate is determined by the synthesizer's default speaking rate; passing NULL means the speaking rate is automatically set to the rate the user specifies in Speech preferences.

chan

On return, a pointer to a valid speech channel.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The NewSpeechChannel function allocates memory for a speech channel structure and sets the speech channel variable pointed to by the chan parameter to point to this speech channel structure. The Speech Synthesis Manager automatically locates and opens a connection to the proper synthesizer for the voice specified by the voice parameter.

There is no predefined limit to the number of speech channels an application can create. However, system constraints on available RAM, processor loading, and number of available sound channels limit the number of speech channels actually possible.

Your application should not attempt to manipulate the data pointed to by a variable of type SpeechChannel. The internal format that the Speech Synthesis Manager uses for speech channel data is not documented and may change in future versions of system software.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechSynthesis.h

NewSpeechDoneUPP

Creates a new universal procedure pointer (UPP) to a speech-done callback function.

```
SpeechDoneUPP NewSpeechDoneUPP (
   SpeechDoneProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your speech-done callback function.

Return Value

A UPP to the speech-done callback function. See the description of the `SpeechDoneUPP` data type.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

NewSpeechErrorUPP

Creates a new universal procedure pointer to an error callback function.

```
SpeechErrorUPP NewSpeechErrorUPP (
   SpeechErrorProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your error callback function.

Return Value

A UPP to the error callback function. See the description of the `SpeechErrorUPP` data type.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

NewSpeechPhonemeUPP

Disposes of a universal procedure pointer (UPP) to a phoneme callback function.

```
SpeechPhonemeUPP NewSpeechPhonemeUPP (
   SpeechPhonemeProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your phoneme callback function.

Return Value

A UPP to the phoneme callback function. See the description of the `SpeechPhonemeUPP` data type.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechSyncUPP

Creates a new universal procedure pointer (UPP) to a synchronization callback function.

```
SpeechSyncUPP NewSpeechSyncUPP (
   SpeechSyncProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your synchronization callback function.

Return Value
A UPP to the synchronization callback function. See the description of the `SpeechSyncUPP` data type.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechTextDoneUPP

Creates a new universal procedure pointer (UPP) to a text-done callback function.

```
SpeechTextDoneUPP NewSpeechTextDoneUPP (
   SpeechTextDoneProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your text-done callback function.

Return Value
A UPP to the text-done callback function. See the description of the `SpeechTextDoneUPP` data type.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

NewSpeechWordUPP

Creates a new universal procedure pointer (UPP) to a word callback function.

```
SpeechWordUPP NewSpeechWordUPP (
   SpeechWordProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your word callback function.

Return Value

A UPP to the word callback function. See the description of the `SpeechWordUPP` data type.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

PauseSpeechAt

Pauses speech on a speech channel.

```
OSErr PauseSpeechAt (
   SpeechChannel chan,
   SInt32 whereToPause
);
```

Parameters

chan

The speech channel on which speech is to be paused.

whereToPause

A constant indicating when speech processing should be paused. Pass the constant `kImmediate` to pause immediately, even in the middle of a word. Pass `kEndOfWord` or `kEndOfSentence` to pause speech at the end of the current word or sentence, respectively.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion

The `PauseSpeechAt` function makes speech production pause at a specified point in the text. `PauseSpeechAt` returns immediately, although speech output will continue until the specified point.

You can determine whether your application has paused speech output on a speech channel by obtaining a speech status information structure through the `GetSpeechInfo` function. While a speech channel is paused, the speech status information structure indicates that `outputBusy` and `outputPaused` are both `TRUE`.

If the end of the input text buffer is reached before the specified pause point, speech output pauses at the end of the buffer.

The `PauseSpeechAt` function differs from the `StopSpeech` and `StopSpeechAt` functions in that a subsequent call to `ContinueSpeech`, described next, causes the contents of the current text buffer to continue being spoken.

If you plan to continue speech synthesis from a paused speech channel, the text buffer being processed must remain available at all times and must not move while the channel is in a paused state.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SetSpeechInfo

Changes a setting of a particular speech channel.

```
OSErr SetSpeechInfo (
   SpeechChannel chan,
   OSType selector,
   const void *speechInfo
);
```

Parameters

chan
The speech channel for which your application wishes to change a setting.

selector
A speech information selector that indicates the type of information being changed.

For a complete list of speech information selectors, see "Speech-Channel Information Constants" (page 1685). This list indicates how your application should set the `speechInfo` parameter for each selector.

speechInfo
A pointer whose meaning depends on the speech information selector specified in the `selector` parameter.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `SetSpeechInfo` function changes the type of setting indicated by the `selector` parameter in the speech channel specified by the `chan` parameter, based on the data your application provides via the `speechInfo` parameter.

The format of the data structure specified by the `speechInfo` parameter depends on the selector you choose. Ordinarily, a selector requires that `speechInfo` be a pointer to a data structure that specifies a new setting for the speech channel.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SetSpeechPitch

Sets the speech pitch on a designated speech channel.

```
OSErr SetSpeechPitch (
   SpeechChannel chan,
   Fixed pitch
);
```

Parameters

chan

The speech channel whose pitch you wish to set.

pitch

The new pitch for the speech channel, expressed as a fixed-point frequency value.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion

The `SetSpeechPitch` function changes the current speech pitch on the speech channel specified by the `chan` parameter to the pitch specified by the `pitch` parameter. Typical voice frequencies range from around 90 hertz for a low-pitched male voice to perhaps 300 hertz for a high-pitched child's voice. These frequencies correspond to approximate pitch values in the ranges of 30.000 to 40.000 and 55.000 to 65.000, respectively. Although fixed-point values allow you to specify a wide range of pitches, not all synthesizers will support the full range of pitches. If your application specifies a pitch that a synthesizer cannot handle, it may adjust the pitch to fit within an acceptable range.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

SetSpeechProperty

Sets the value of the specified speech-channel property.

```
OSErr SetSpeechProperty (
   SpeechChannel chan,
   CFStringRef property,
   CFTypeRef object
);
```

Parameters

chan

The speech channel whose property to set.

property

The speech-channel property to set to the specified value.

object

The value to which the specified speech-channel property should be set. For some properties, this value can be `NULL`.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `SetSpeechProperty` function is the Core Foundation-based equivalent of the `SetSpeechInfo` (page 1650) function.

See "`Speech-Channel Properties`" (page 1692) for information on the properties you can specify.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

SetSpeechRate

Sets the speech rate of a designated speech channel.

```
OSErr SetSpeechRate (
   SpeechChannel chan,
   Fixed rate
);
```

Parameters

`chan`
The speech channel whose rate you wish to set.

`rate`
The new speech rate in words per minute, expressed as an integer value.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `SetSpeechRate` function adjusts the speech rate on the speech channel specified by the `chan` parameter to the rate specified by the `rate` parameter. As a general rule, speaking rates range from around 150 words per minute to around 220 words per minute. It is important to keep in mind, however, that users will differ greatly in their ability to understand synthesized speech at a particular rate based upon their level of experience listening to the voice and their ability to anticipate the types of utterances they will encounter.

Note: the new speech rate should be expressed as an integer (not a fixed point decimal number as the data type implies).

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeakBuffer

Speaks a buffer of text, using certain flags to control speech behavior.

```
OSErr SpeakBuffer (
   SpeechChannel chan,
   const void *textBuf,
   unsigned long textBytes,
   SInt32 controlFlags
);
```

Parameters

chan

The speech channel through which speech is to be spoken.

textBuf

A pointer to the first byte of text to spoken.

textBytes

The number of bytes of text to spoken.

controlFlags

Control flags to customize speech behavior.

Return Value

A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion

The `SpeakBuffer` function behaves identically to the `SpeakText` function, but allows control of several speech parameters by setting values of the `controlFlags` parameter. The `controlFlags` parameter relies on specific constants, which may be applied additively. See "Control Flags Constants" (page 1680).

Each constant specifies a flag bit of the `controlFlags` parameter, so by passing the constants additively you can enable multiple capabilities of `SpeakBuffer`. If you pass `0` in the `controlFlags` parameter, `SpeakBuffer` works just like `SpeakText`. By passing `kNoEndingProsody + kNoSpeechInterrupt` in the `controlFlags` parameter, `SpeakBuffer` works like `SpeakText` except that the `kNoEndingProsody` and `kNoSpeechInterrupt` features have been selected. Future versions of the Speech Synthesis Manager may define additional constants.

When the `controlFlags` parameter is set to `0`, `SpeakBuffer` behaves identically to `SpeakText`.

Availability

Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeakCFString

Begins speaking a string represented as a `CFString` object.

```
OSErr SpeakCFString (
   SpeechChannel chan,
   CFStringRef aString,
   CFDictionaryRef options
);
```

Parameters

chan

The speech channel through which speech is to be spoken.

aString

The string to be spoken, represented as a `CFString` object.

options

An optional dictionary of key-value pairs used to customize speech behavior. See "`Synthesizer Option Keys`" (page 1698) for the available keys.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `SpeakCFString` function is the Core Foundation-based equivalent of the `SpeakBuffer` (page 1652) function.

The `SpeakCFString` function converts the text string specified in *aString* into speech, using the voice and control settings in effect for the speech channel specified in *chan*. (Before you use `SpeakCFString`, therefore, be sure you've created a speech channel with the `NewSpeechChannel` (page 1646) function.) The `SpeakCFString` function generates speech asynchronously, which means that control is returned to your application before speech has finished, perhaps even before the speech is first audible.

If `SpeakCFString` is called while the speech channel is currently speaking the contents of another text string, the speech stops immediately and the new text string is spoken as soon as possible.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

SpeakString

Begins speaking a text string.

```
OSErr SpeakString (
   ConstStr255Param textToBeSpoken
);
```

Parameters

textToBeSpoken

The string to be spoken.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `SpeakString` function attempts to speak the Pascal-style text string contained in the string `textToBeSpoken`. Speech is produced asynchronously using the default system voice. When an application calls this function, the Speech Synthesis Manager makes a copy of the passed string and creates any structures required to speak it. As soon as speaking has begun, control is returned to the application. The synthesized speech is generated asynchronously to the application so that normal processing can continue while the text is being spoken. No further interaction with the Speech Synthesis Manager is required at this point, and the application is free to release the memory that the original string occupied.

If `SpeakString` is called while a prior string is still being spoken, the sound currently being synthesized is interrupted immediately. Conversion of the new text into speech is then begun. If you pass a zero-length string (or, in C, a `null` pointer) to `SpeakString`, the Speech Synthesis Manager stops any speech previously being synthesized by `SpeakString` without generating additional speech. If your application uses `SpeakString`, it is often a good idea to stop any speech in progress whenever your application receives a suspend event. Calling `SpeakString` with a zero-length string has no effect on speech channels other than the one managed internally by the Speech Synthesis Manager for the `SpeakString` function.)

The text passed to the `SpeakString` function may contain embedded speech commands.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeakText

Begins speaking a buffer of text.

```
OSErr SpeakText (
   SpeechChannel chan,
   const void *textBuf,
   unsigned long textBytes
);
```

Parameters

chan
The speech channel through which speech is to be spoken.

textBuf
A pointer to the first byte of text to spoken.

textBytes
The number of bytes of text to spoken.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
Like `SpeakString`, the `SpeakText` function also generates speech, but through a speech channel through which you can exert control over the generated speech.

The SpeakText function converts the text stream specified by the textBuf and textBytes parameters into speech using the voice and control settings for the speech channel chan, which should be created with the NewSpeechChannel function. The speech is generated asynchronously. This means that control is returned to your application before the speech has finished (and probably even before it has begun). The maximum length of the text buffer that can be spoken is limited only by the available RAM.

If SpeakText is called while the channel is currently busy speaking the contents of a prior text buffer, it immediately stops speaking from the prior buffer and begins speaking from the new text buffer as soon as possible. If you pass a zero-length string (or, in C, a null pointer) to SpeakText, the Speech Synthesis Manager stops all speech currently being synthesized by the speech channel specified in the chan parameter without generating additional speech.

The text buffer must be locked in memory and must not move while the Speech Synthesis Manager processes it. This buffer is read at interrupt time, and moving it could cause a system crash. If your application defines a text-done callback function, then it can move the text buffer or dispose of it once the callback function is executed.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechSynthesis.h

SpeechBusy

Determines whether any channels of speech are currently synthesizing speech.

```
SInt16 SpeechBusy (
   void
);
```

Return Value
The number of speech channels that are currently synthesizing speech in the application. This is useful when you want to ensure that an earlier speech request has been completed before having the system speak again. Paused speech channels are counted among those that are synthesizing speech.

The speech channel that the Speech Synthesis Manager allocates internally in response to calls to the SpeakString function is counted in the number returned by SpeechBusy. Thus, if you use just SpeakString to initiate speech, SpeechBusy always returns 1 as long as speech is being produced. When SpeechBusy returns 0, all speech has finished.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechSynthesis.h

SpeechBusySystemWide

Determines if any speech is currently being synthesized in your application or elsewhere on the computer.

```
SInt16 SpeechBusySystemWide (
   void
);
```

Return Value
The total number of speech channels currently synthesizing speech on the computer, whether they were initiated by your application or process's code or by some other process executing concurrently. Paused speech channels are counted among those channels that are synthesizing speech.

Discussion
This function is useful when you want to ensure that no speech is currently being produced anywhere on the Macintosh computer before initiating speech. Although the Speech Synthesis Manager allows different applications to produce speech simultaneously, this can be confusing to the user. As a result, it is often a good idea for your application to check that no other process is producing speech before producing speech itself. If the difference between the values returned by `SpeechBusySystemWide` and the `SpeechBusy` function is `0`, no other process is producing speech.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechManagerVersion

Determines the current version of the Speech Synthesis Manager installed in the system.

```
NumVersion SpeechManagerVersion (
   void
);
```

Return Value
The version of the Speech Synthesis Manager installed in the system, in the format of the first 4 bytes of a `'vers'` resource.

Discussion
Use this call to determine whether your program can access features of the Speech Synthesis Manager that are included in some Speech Synthesis Manager releases but not in earlier ones.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

StopSpeech

Terminates speech immediately on the specified channel.

```
OSErr StopSpeech (
   SpeechChannel chan
);
```

Parameters

chan

The speech channel on which speech is to be stopped.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `StopSpeech` function immediately terminates speech on the channel specified by the `chan` parameter. After returning from `StopSpeech`, your application can safely release any text buffer that the speech synthesizer has been using. You can call `StopSpeech` for an already idle channel without ill effect.

You can also stop speech by passing a zero-length string (or, in C, a `null` pointer) to one of the `SpeakString`, `SpeakText`, or `SpeakBuffer` functions. Doing this stops speech only in the specified speech channel (or, in the case of `SpeakString`, in the speech channel managed internally by the Speech Synthesis Manager).

Before calling the `StopSpeech` function, you can use the `SpeechBusy` function, which is described in `SpeechBusy` (page 1656), to determine if a synthesizer is still speaking. If you are working with multiple speech channels, you can use the status selector with the function `GetSpeechInfo` which is described in `GetSpeechInfo` (page 1638), to determine if a specific channel is still speaking.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

StopSpeechAt

Terminates speech delivery on a specified channel either immediately or at the end of the current word or sentence.

```
OSErr StopSpeechAt (
   SpeechChannel chan,
   SInt32 whereToStop
);
```

Parameters

chan

The speech channel on which speech is to be stopped.

whereToStop

A constant indicating when speech processing should stop. Pass the constant `kImmediate` to stop immediately, even in the middle of a word. Pass `kEndOfWord` or `kEndOfSentence` to stop speech at the end of the current word or sentence, respectively.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `StopSpeechAt` function halts the production of speech on the channel specified by `chan` at a specified point in the text. This function returns immediately, although speech output continues until the specified point has been reached.

If you call the `StopSpeechAt` function before the Speech Synthesis Manager finishes processing input text, then the function might return before some input text has yet to be spoken. Thus, before disposing of the text buffer, your application should wait until its text-done callback function has been called (if one has been defined), or until it can determine (by, for example obtaining a speech status information structure) that the Speech Synthesis Manager is no longer processing input text.

If the end of the input text buffer is reached before the specified stopping point, the speech synthesizer stops at the end of the buffer without generating an error.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

TextToPhonemes

Converts a buffer of textual data into phonemic data.

```
OSErr TextToPhonemes (
   SpeechChannel chan,
   const void *textBuf,
   unsigned long textBytes,
   Handle phonemeBuf,
   long *phonemeBytes
);
```

Parameters

chan
A speech channel whose associated synthesizer and voice are to be used for the conversion process.

textBuf
A pointer to a buffer of text to be converted.

textBytes
The number of bytes of text to be converted.

phonemeBuf
A handle to a buffer to be used to store the phonemic data. The `TextToPhonemes` function may resize the relocatable block referenced by this handle.

phonemeBytes
On return, a pointer to the number of bytes of phonemic data written to the handle.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
Converting textual data into phonemic data is particularly useful during application development, when you might wish to adjust phrases that your application generates to produce smoother speech. By first converting the target phrase into phonemes, you can see what the synthesizer will try to speak. Then you need correct only the parts that would not have been spoken the way you want.

The `TextToPhonemes` function converts the `textBytes` bytes of textual data pointed to by the `textBuf` parameter to phonemic data, which it writes into the relocatable block specified by the `phonemeBuf` parameter. If necessary, `TextToPhonemes` resizes this relocatable block. The `TextToPhonemes` function sets the `phonemeBytes` parameter to the number of bytes of phonemic data actually written.

If the textual data is contained in a relocatable block, a handle to that block must be locked before the `TextToPhonemes` function is called.

The data returned by `TextToPhonemes` corresponds precisely to the phonemes that would be spoken had the input text been sent to `SpeakText` instead. All current mode settings for the speech channel specified by `chan` are applied to the converted speech. No callbacks are generated while the `TextToPhonemes` function is generating its output.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

UseDictionary

Installs the designated dictionary into a speech channel.

```
OSErr UseDictionary (
   SpeechChannel chan,
   Handle dictionary
);
```

Parameters

chan
The speech channel into which a dictionary is to be installed.

dictionary
A handle to the dictionary data. This is often a handle to a resource of type `'dict'`.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The `UseDictionary` function attempts to install the dictionary data referenced by the `dictionary` parameter into the speech channel referenced by the `chan` parameter. The synthesizer will use whatever elements of the dictionary resource it considers useful to the speech conversion process. Some speech synthesizers might ignore certain types of dictionary entries.

After the `UseDictionary` function returns, your application is free to release any storage allocated for the dictionary handle. The search order for application-provided dictionaries is last-in, first-searched.

All details of how an application-provided dictionary is represented within the speech synthesizer are dependent on the specific synthesizer implementation and are private to the synthesizer.

Pronunciation dictionaries allow your application to override the default Speech Synthesis Manager pronunciations of individual words, such as names with unusual spellings.

Availability
Available in CarbonLib 1.0 and later when Text to Speech 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechSynthesis.h`

UseSpeechDictionary

Registers a speech dictionary with a speech channel.

```
OSErr UseSpeechDictionary (
   SpeechChannel chan,
   CFDictionaryRef speechDictionary
);
```

Parameters

chan
The speech channel with which the specified speech dictionary is to be registered.

speechDictionary
A speech dictionary to be registered with the specified speech channel, represented as a `CFDictionary` object. See "Speech Dictionary Keys" (page 1704) for the keys you can use in the dictionary.

Return Value
A result code. See "Speech Synthesis Manager Result Codes" (page 1705).

Discussion
The UseSpeechDictionary function is the Core Foundation-based equivalent of the `UseDictionary` (page 1660) function.

The `UseSpeechDictionary` function registers the `CFDictionary` object referenced by the `speechDictionary` parameter with the speech channel referenced by the `chan` parameter. Speech dictionaries allow your application to override a synthesizer's default pronunciations of individual words, such as names with unusual spellings. A synthesizer will use whatever elements of the dictionary it considers useful in the speech conversion process. Some speech synthesizers might ignore certain types of dictionary entries.

Multiple dictionaries can be registered with a synthesizer. If the same word appears in multiple dictionaries, the synthesizer will use the one from the dictionary with the most recent date.

Note that because a speech dictionary is a `CFDictionary` object, it can be loaded from an XML-based property list file. An example of such a file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
```

```
    <key>LocaleIdentifier</key>
    <string>en_US</string>
    <key>ModificationDate</key>
    <string>2006-12-21 11:59:25 -0800</string>
    <key>Pronunciations</key>
    <array>
        <dict>
            <key>Phonemes</key>
            <string>_hEY_yUW</string>
            <key>Spelling</key>
            <string>Hello</string>
        </dict>
    </array>
    <key>Abbreviations</key>
    <array>
        <dict>
            <key>Phonemes</key>
            <string>_OW_sAEkz</string>
            <key>Spelling</key>
            <string>OSAX</string>
        </dict>
    </array>
</dict>
</plist>
```

After the `UseSpeechDictionary` function returns, your application is free to release the `CFDictionary` object referenced by the `speechDictionary` parameter.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

Callbacks

SpeechDoneProcPtr

Defines a pointer to a speech-done callback function which is called when the Speech Synthesis Manager finishes speaking a buffer of text.

```
typedef void (*SpeechDoneProcPtr) (
    SpeechChannel chan,
    SRefCon refCon
);
```

If you name your function `MySpeechDoneProc`, you would declare it like this:

```
void MySpeechDoneProc (
    SpeechChannel chan,
    long refCon
);
```

Parameters

chan

The speech channel that has finished processing input text.

refCon

The reference constant associated with the speech channel.

Discussion

If a speech-done callback function is installed in a speech channel, then the Speech Synthesis Manager calls this function when it finishes speaking a buffer of text.

You can specify a speech-done callback function by passing the `soSpeechDoneCallBack` selector to the `SetSpeechInfo` function.

You might use a speech-done callback function if you need to update some visual indicator that shows what text is currently being spoken. For example, suppose your application passes text buffers to the Speech Synthesis Manager one paragraph at a time. Your speech-done callback function might set a global flag variable to indicate to the application that the Speech Synthesis Manager has finished speaking a paragraph. When a function called by your application's main event loop checks the global flag variable and determines that it has been set, the function might ensure that the next paragraph of text is visible.

You might use a speech-done callback function to set a flag variable that alerts the application that it should pass a new buffer of text to the Speech Synthesis Manager. If you do so, however, there might be a noticeable pause as the Speech Synthesis Manager switches from processing one text buffer to another. Ordinarily, it is easier to achieve this goal by using a text-done callback function, as described earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechErrorCFProcPtr

Defines a pointer to an error callback function that handles syntax errors within commands embedded in a `CFString` object being processed by the Speech Synthesis Manager.

```
typedef void (*SpeechErrorCFProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    CFErrorRef theError
);
```

If you name your function `MySpeechErrorCFProc`, you would declare it like this:

```
void MySpeechErrorCFProc (
    SpeechChannel chan,
    long refCon,
    CFErrorRef theError
);
```

Parameters

chan

The speech channel that has finished processing input text.

refCon
The reference constant associated with the speech channel.

theError
The error that occurred in processing an embedded command.

Discussion
An error callback function defined by the `SpeechErrorCFProcPtr` is the Core Foundation-based equivalent of an error callback function defined by `SpeechErrorProcPtr` (page 1664). The Speech Synthesis Manager calls a speech channel's error callback function whenever it encounters a syntax error within a command embedded in a `CFString` object it is processing. This can be useful during application debugging, to detect problems with commands that you have embedded in strings that your application speaks. It can also be useful if your application allows users to embed commands within strings. Your application might display an alert indicating that the Speech Synthesis Manager encountered a problem in processing an embedded command.

Ordinarily, the error information that the Speech Synthesis Manager provides the error callback function should be sufficient. However, if your application needs information about errors that occurred before the error callback function was enabled, the application (including the error callback function) can call the `CopySpeechProperty` (page 1634) function with the `kSpeechErrorsProperty` property.

You can specify an error callback function by passing the `kSpeechErrorCFCallback` property to the `SetSpeechProperty` (page 1651) function.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

SpeechErrorProcPtr

Defines a pointer to an error callback function that handles syntax errors within commands embedded in a text buffer being processed by the Speech Synthesis Manager.

```
typedef void (*SpeechErrorProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    OSErr theError,
    long bytePos
);
```

If you name your function `MySpeechErrorProc`, you would declare it like this:

```
void MySpeechErrorProc (
    SpeechChannel chan,
    long refCon,
    OSErr theError,
    long bytePos
);
```

Parameters

chan
The speech channel that has finished processing input text.

refCon
The reference constant associated with the speech channel.

theError
The error that occurred in processing an embedded command.

bytePos
The number of bytes from the beginning of the text buffer being spoken to the error encountered.

Discussion
The Speech Synthesis Manager calls a speech channel's error callback function whenever it encounters a syntax error within a command embedded in a text buffer it is processing. This can be useful during application debugging, to detect problems with commands that you have embedded in text buffers that your application speaks. It can also be useful if your application allows users to embed commands within text buffers. Your application might display an alert indicating that the Speech Synthesis Manager encountered a problem in processing an embedded command.

Ordinarily, the error information that the Speech Synthesis Manager provides the error callback function should be sufficient. However, if your application needs information about errors that occurred before the error callback function was enabled, the application (including the error callback function) can call the `GetSpeechInfo` function with the `soErrors` selector.

You can specify an error callback function by passing the `soErrorCallBack` selector to the `SetSpeechInfo` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechPhonemeProcPtr

Defines a pointer to a phoneme callback function that is called by the Speech Synthesis Manager before it pronounces a phoneme.

```
typedef void (*SpeechPhonemeProcPtr)
(
    SpeechChannel chan,
    SRefCon refCon,
    short phonemeOpcode
);
```

If you name your function `MySpeechPhonemeProc`, you would declare it like this:

```
void MySpeechPhonemeProc (
    SpeechChannel chan,
    long refCon,
    short phonemeOpcode
);
```

Parameters

chan
The speech channel that has finished processing input text.

refCon
The reference constant associated with the speech channel.

phonemeOpcode
The phoneme about to be pronounced.

Discussion
The Speech Synthesis Manager calls a speech channel's phoneme callback function just before it pronounces a phoneme. For example, your application might use such a callback function to enable mouth synchronization. In this case, the callback function would set a global flag variable to indicate that the phoneme being pronounced is changing and another global variable to `phonemeOpcode`. A function called by your application's main event loop could detect that the phoneme being pronounced is changing and update a picture of a mouth to reflect the current phoneme. In practice, providing a visual indication of the pronunciation of a phoneme requires several consecutive pictures of mouth movement to be rapidly displayed. Consult the linguistics literature for information on mouth movements associated with different phonemes.

You can specify a phoneme callback function by passing the `soPhonemeCallBack` selector to the `SetSpeechInfo` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechSyncProcPtr

Defines a pointer to a synchronization callback function that is called when the Speech Synthesis Manager encounters a synchronization command embedded in a text buffer.

```
typedef void (*SpeechSyncProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    OSType syncMessage
);
```

If you name your function `MySpeechSyncProc`, you would declare it like this:

```
void MySpeechSyncProc (
    SpeechChannel chan,
    long refCon,
    OSType syncMessage
);
```

Parameters

chan
The speech channel that has finished processing input text.

refCon
The reference constant associated with the speech channel.

syncMessage
The synchronization message passed in the embedded command. Usually, you use this message to distinguish between several different types of synchronization commands, but you can use it any way you wish.

Discussion
The Speech Synthesis Manager calls a speech channel's synchronization callback function whenever it encounters a synchronization command embedded in a text buffer. You might use the synchronization callback function to provide a callback not ordinarily provided. For example, you might inset synchronization commands at the end of every sentence in a text buffer, or you might enter synchronization commands after every numeric value in the text. However, to synchronize your application with phonemes or words, it makes more sense to use the built-in phoneme and word callback functions, defined in `SpeechPhonemeProcPtr` (page 1665).

You can specify a synchronization callback function by passing the `soSyncCallBack` selector to the `SetSpeechInfo` function and embedding a synchronization command within a text buffer passed to the `SpeakText` or `SpeakBuffer` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechTextDoneProcPtr

Defines a pointer to a text-done callback function that is called when the Speech Synthesis Manager has finished processing a buffer of text.

```
typedef void (*SpeechTextDoneProcPtr)
(
    SpeechChannel chan,
    SRefCon refCon,
    void ** nextBuf,
    unsigned long * byteLen,
    long * controlFlags
);
```

If you name your function `MySpeechTextDoneProc`, you would declare it like this:

```
void MySpeechTextDoneProc (
    SpeechChannel chan,
    long refCon,
    void ** nextBuf,
    unsigned long * byteLen,
    long * controlFlags
);
```

Parameters

chan
The speech channel that has finished processing input text.

refCon
The reference constant associated with the speech channel.

nextBuf
On return, a pointer to the next buffer of text to process or `NULL` if your application has no additional text to be spoken. This parameter is mostly for internal use by the Speech Synthesis Manager.

byteLen

On return, a pointer to the number of bytes of the text buffer pointed to by the `nextBuf` parameter.

controlFlags

On return, a pointer to the control flags to be used in generating the next buffer of text.

Discussion

If a text-done callback function is installed in a speech channel, then the Speech Synthesis Manager calls this function when it finishes processing a buffer of text. The Speech Synthesis Manager might not yet have completed finishing speaking the text and indeed might not have started speaking it.

You can specify a text-done callback function by passing the `soTextDoneCallBack` selector to the `SetSpeechInfo` function.

A common use of a text-done callback function is to alert your application once the text passed to the `SpeakText` or `SpeakBuffer` function can be disposed of (or, when the text is contained within a locked relocatable block, when the relocatable block can be unlocked). The Speech Synthesis Manager copies the text you pass to the `SpeakText` or `SpeakBuffer` function into an internal buffer. Once it has finished processing the text, you may dispose of the original text buffer, even if speech is not yet complete. However, if you wish to write a callback function that executes when speech is completed, see the definition of a speech-done callback function below.

Although most applications will not need to, your callback function can indicate to the Speech Synthesis Manager whether there is another buffer of text to speak. If there is another buffer, your callback function should reference it by setting the `nextBuf` and `byteLen` parameters to appropriate values. (Your callback function might also change the control flags to be used to process the speech by altering the value in the `controlFlags` parameter.) Setting these parameters allows the Speech Synthesis Manager to generate uninterrupted speech. If there is no more text to speak, your callback function should set `nextBuf` to `NULL`. In this case, the Speech Synthesis Manager ignores the `byteLen` and `controlFlags` parameters.

If your text-done callback function does not change the values of the `nextBuf` and `byteLen` parameters, the text buffer just spoken will be spoken again.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechWordCFProcPtr

Defines a pointer to a Core Foundation-based word callback function that is called by the Speech Synthesis Manager before it pronounces a word.

```
typedef void (*SpeechWordCFProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    CFStringRef aString,
    CFRange wordRange
);
```

If you name your function `MySpeechWordCFProc`, you would declare it like this:

```
void MySpeechWordCFProc (
    SpeechChannel chan,
    SRefCon refCon,
```

```
    CFStringRef aString,
    CFRange wordRange
);
```

Parameters

chan

The speech channel that has finished processing input text.

refCon

The reference constant associated with the speech channel.

aString

The original string passed to the speech synthesizer in the `SpeakCFString` (page 1653) call.

wordRange

The range of characters in `aString` that corresponds to the word.

Discussion

A word callback function defined by the `SpeechWordCFProcPtr` is the Core Foundation-based equivalent of a word callback function defined by `SpeechWordProcPtr` (page 1669). The Speech Synthesis Manager calls a speech channel's word callback function just before it pronounces a word. You might use such a callback function, for example, to highlight the word about to be spoken in a window.

You can specify a word callback function by passing the `kSpeechWordCFCallBack` property to the`SetSpeechProperty` (page 1651) function.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SpeechSynthesis.h`

SpeechWordProcPtr

Defines a pointer to a word callback function that is called by the Speech Synthesis Manager before it pronounces a word.

```
typedef void (*SpeechWordProcPtr) (
    SpeechChannel chan,
    SRefCon refCon,
    unsigned long wordPos,
    unsigned short wordLen
);
```

If you name your function `MySpeechWordProc`, you would declare it like this:

```
void MySpeechWordProc (
    SpeechChannel chan,
    long refCon,
    unsigned long wordPos,
    unsigned short wordLen
);
```

Parameters

chan

The speech channel that has finished processing input text.

refCon

The reference constant associated with the speech channel.

wordPos

The number of bytes between the beginning of the text buffer and the beginning of the word about to be pronounced.

wordLen

The length in bytes of the word about to be pronounced.

Discussion

The Speech Synthesis Manager calls a speech channel's word callback function just before it pronounces a word. You might use such a callback function, for example, to draw the word about to be spoken in a window. In this case, the callback function would set a global flag variable to indicate that the word being spoken is changing and another two global variables to `wordPos` and `wordLen`. A function called by your application's main event loop could detect that the word being spoken is changing and draw the word in a window.

You can specify a word callback function by passing the `soWordCallBack` selector to the `SetSpeechInfo` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

Data Types

DelimiterInfo

Defines a delimiter information structure.

```
struct DelimiterInfo {
    Byte startDelimiter[2];
    Byte endDelimiter[2];
};
typedef struct DelimiterInfo DelimiterInfo;
```

Fields

`startDelimiter`

The start delimiter for an embedded command. By default, the start delimiter is "`[[`".

`endDelimiter`

The end delimiter for an embedded command. By default, the end delimiter is "`]]`".

Discussion

A delimiter information structure defines the characters used to indicate the beginning and end of a command embedded in text. A delimiter can be one or two characters.

Ordinarily, applications that support embedded speech commands should not change the start or end delimiters. However, if for some reason you must change the delimiters, you can use the `SetSpeechInfo` function with the `soCommandDelimiter` selector. For example, you might do this if a text buffer naturally includes the delimiter strings. Before passing such a buffer to the Speech Synthesis Manager, you can change the delimiter strings to some two-character sequences not used in the buffer and then change the delimiter strings back once processing of the buffer is complete.

If a single-byte delimiter is desired, it should be followed by a `NULL` (0) byte. If the delimiter strings both consist of two `NULL` bytes, embedded command processing is disabled.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

PhonemeDescriptor

Defines a phoneme descriptor structure.

```
struct PhonemeDescriptor {
    SInt16 phonemeCount;
    PhonemeInfo thePhonemes[1];
};
typedef struct PhonemeDescriptor PhonemeDescriptor;
```

Fields

`phonemeCount`
The number of phonemes that the current synthesizer defines. Typically, this will correspond to the number of phonemes in the language supported by the synthesizer.

`thePhonemes`
An array of phoneme information structures.

Discussion
By calling the `GetSpeechInfo` (page 1638) function with the `soPhonemeSymbols` selector, you can obtain a phoneme descriptor structure, which describes all phonemes defined for the current synthesizer.

A common use for a phoneme descriptor structure is to provide a graphical display to the user of all available phonemes. Such a list is used only for a user entering phonemic data directly rather than just entering text.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

PhonemeInfo

Defines a structure that stores information about a phoneme.

```
struct PhonemeInfo {
    SInt16 opcode;
    Str15 phStr;
    Str31 exampleStr;
    SInt16 hiliteStart;
    SInt16 hiliteEnd;
};
typedef struct PhonemeInfo PhonemeInfo;
```

Fields

`opcode`
The opcode for the phoneme.

`phStr`
The string used to represent the phoneme. The string does not necessarily have a phonetic connection to the phoneme, but might simply be an abstract textual representation of it.

`exampleStr`
An example word that illustrates use of the phoneme.

`hiliteStart`
The number of characters in the example word that precede the portion of that word representing the phoneme.

`hiliteEnd`
The number of characters between the beginning of the example word and the end of the portion of that word representing the phoneme.

Discussion

Ordinarily, you use a phoneme information structure to show the user how to enter text to represent a particular phoneme when the `'PHON'` input mode is activated.

You might use the information contained in the `hiliteStart` and `hiliteEnd` fields to highlight the characters in the example word that represent the phoneme.

To obtain a phoneme information structure for an individual phoneme, you must obtain a list of phonemes through a phoneme descriptor structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechChannelRecord

Represents a speech channel.

```
struct SpeechChannelRecord {
    long data[1];
};
typedef struct SpeechChannelRecord SpeechChannelRecord;
typedef SpeechChannelRecord * SpeechChannel;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
SpeechSynthesis.h

SpeechDoneUPP

Defines a universal procedure pointer (UPP) to a speech-done callback function.

```
typedef SpeechDoneProcPtr SpeechDoneUPP;
```

Discussion
For more information, see the description of the SpeechDoneProcPtr (page 1662) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechSynthesis.h

SpeechErrorInfo

Defines a speech error information structure.

```
struct SpeechErrorInfo {
    SInt16 count;
    OSErr oldest;
    long oldPos;
    OSErr newest;
    long newPos;
};
typedef struct SpeechErrorInfo SpeechErrorInfo;
```

Fields

count
: The number of errors that have occurred in processing the current text buffer since the last call to the GetSpeechInfo function with the soErrors selector. Of these errors, you can find information about only the first and last error that occurred.

oldest
: The error code of the first error that occurred after the previous call to the GetSpeechInfo function with the soErrors selector.

oldPos
: The character position within the text buffer being processed of the first error that occurred after the previous call to the GetSpeechInfo function with the soErrors selector.

newest
: The error code of the most recent error.

newPos
: The character position within the text buffer being processed of the most recent error.

Discussion
By calling the GetSpeechInfo (page 1638) function with the soErrors selector, you can obtain a speech error information structure, which shows what Speech Synthesis Manager errors occurred while processing a text buffer on a given speech channel.

Speech error information structures never include errors that are returned by Speech Synthesis Manager functions. Instead, they reflect only errors encountered directly in the processing of text, and, in particular, in the processing of commands embedded within text.

The speech error information structure keeps track of only the most recent error and the first error that occurred after the previous call to the `GetSpeechInfo` function with the `soErrors` selector. If your application needs to keep track of all errors, then you should install an error callback function, `SpeechErrorProcPtr` (page 1664).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechErrorUPP

Defines a universal procedure pointer (UPP) to an error callback function.

```
typedef SpeechErrorProcPtr SpeechErrorUPP;
```

Discussion
For more information, see the description of the `SpeechErrorProcPtr` (page 1664) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechPhonemeUPP

Defines a universal procedure pointer (UPP) to a phoneme callback function.

```
typedef SpeechPhonemeProcPtr SpeechPhonemeUPP;
```

Discussion
For more information, see the description of the `SpeechPhonemeProcPtr` (page 1665) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechStatusInfo

Defines a a speech status information structure, which stores information about the status of a speech channel.

```
struct SpeechStatusInfo {
    Boolean outputBusy;
    Boolean outputPaused;
    long inputBytesLeft;
    SInt16 phonemeCode;
};
typedef struct SpeechStatusInfo SpeechStatusInfo;
```

Fields

`outputBusy`

Whether the speech channel is currently producing speech. A speech channel is considered to be producing speech even at some times when no audio data is being produced through the Macintosh speaker. This occurs, for example, when the Speech Synthesis Manager is processing an input buffer but has not yet initiated speech or when speech output is paused.

`outputPaused`

Whether speech output in the speech channel has been paused by a call to the `PauseSpeechAt` (page 1649) function.

`inputBytesLeft`

The number of input bytes of the text that the speech channel must still process. When `inputBytesLeft` is 0, the buffer of input text passed to one of the `SpeakText` or `SpeakBuffer` functions may be disposed of. When you call the `SpeakString` function, the Speech Synthesis Manager stores a duplicate of the string to be spoken in an internal buffer; thus, you may delete the original string immediately after calling `SpeakString`.

`phonemeCode`

The opcode for the phoneme that the speech channel is currently processing.

Discussion

By calling the `GetSpeechInfo` (page 1638) function with the `soStatus` selector, you can find out information about the status of a speech channel.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechSyncUPP

Defines a universal procedure pointer (UPP) to a synchronization callback function.

```
typedef SpeechSyncProcPtr SpeechSyncUPP;
```

Discussion

For more information, see the description of the `SpeechSyncProcPtr` (page 1666) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechSynthesis.h`

SpeechTextDoneUPP

Defines a universal procedure pointer (UPP) to a text-done callback function.

```
typedef SpeechTextDoneProcPtr SpeechTextDoneUPP;
```

Discussion
For more information, see the description of the `SpeechTextDoneProcPtr` (page 1667) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

SpeechVersionInfo

Defines a speech version information structure.

```
struct SpeechVersionInfo {
    OSType synthType;
    OSType synthSubType;
    OSType synthManufacturer;
    SInt32 synthFlags;
    NumVersion synthVersion;
};
typedef struct SpeechVersionInfo SpeechVersionInfo;
```

Fields

`synthType`
The general type of the synthesizer. For the current version of the Speech Synthesis Manager, this field always contains the value `kTextToSpeechSynthType`, indicating that the synthesizer converts text into speech.

`synthSubType`
The specific type of the synthesizer. Currently, no specific types of synthesizer are defined. If you define a new type of synthesizer, you should register the four-character code for your type with Developer Technical Support.

`synthManufacturer`
A unique identification of a synthesizer engine. If you develop synthesizers, then you should register a different four-character code for each synthesizer you develop with Developer Technical Support. The `creatorID` field of the voice specification structure and the `synthCreator` field of a speech extension data structure should each be set to the value stored in this field for the desired synthesizer.

`synthFlags`
A set of flags indicating which synthesizer features are activated. Specific constants define the bits in this field whose meanings are defined for all synthesizers.

`synthVersion`
The version number of the synthesizer.

Discussion
By calling the `GetSpeechInfo` (page 1638) function with the `soSynthType` selector, you can obtain a speech version information structure, which provides information about the speech synthesizer currently being used.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechSynthesis.h

SpeechWordUPP

Defines a universal procedure pointer (UPP) to a word callback function.

```
typedef SpeechWordProcPtr SpeechWordUPP;
```

Discussion
For more information, see the description of the SpeechWordProcPtr (page 1669) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechSynthesis.h

SpeechXtndData

Defines a speech extension data structure.

```
struct SpeechXtndData {
    OSType synthCreator;
    Byte synthData[2];
};
typedef struct SpeechXtndData SpeechXtndData;
```

Fields

synthCreator
: The synthesizer's creator ID, identical to the value stored in the synthManufacturer field of a speech version information structure. You should set this field to the appropriate value before calling GetSpeechInfo or SetSpeechInfo.

synthData
: Synthesizer-specific data. The size and format of the data in this field may vary.

Discussion
The speech extension data structure allows you to use the GetSpeechInfo (page 1638) and SetSpeechInfo (page 1650) functions with selectors defined by particular synthesizers. By requiring that you pass to one of these functions a pointer to a speech extension data structure, synthesizers can permit the exchange of data in any format.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechSynthesis.h

VoiceDescription

Defines a voice description structure.

```
struct VoiceDescription {
    SInt32 length;
    VoiceSpec voice;
    SInt32 version;
    Str63 name;
    Str255 comment;
    SInt16 gender;
    SInt16 age;
    SInt16 script;
    SInt16 language;
    SInt16 region;
    SInt32 reserved[4];
};
typedef struct VoiceDescription VoiceDescription;
```

Fields

`length`
The size of the voice description structure, in bytes.

`voice`
A voice specification structure that uniquely identifies the voice.

`version`
The version number of the voice.

`name`
The name of the voice, preceded by a length byte. Names must be 63 characters or less.

`comment`
Additional text information about the voice. Some synthesizers use this field to store a phrase that can be spoken.

`gender`
The gender of the individual represented by the voice. See "Gender Constants" (page 1681).

`age`
The approximate age in years of the individual represented by the voice.

`script`
In Mac OS X v10.4.7 and later, the encoding code of the text that the voice can process.

Note that this field contains a 16-bit value. You can use any of the 16-bit values described in `External_String_Encodings` or `CFStringBuiltInEncodings`. However, if you need to use a 32-bit value, such as `kCFStringEncodingUTF8`, you pass the value in the first array element of the `reserved` field, and you also specify `-1` or `kCFStringEncodingInvalidId` in the `script` field.

`language`
A code that indicates the language of voice output.

`region`
A code that indicates the region represented by the voice.

`reserved`
Reserved. May be used to hold a 32-bit encoding value, if necessary (see the description of the `script` field for more information).

Discussion

By calling the `GetVoiceDescription` (page 1640) function, you can obtain information about a voice in a voice description structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In
SpeechSynthesis.h

VoiceFileInfo

Defines a voice file information structure.

```
struct VoiceFileInfo {
    FSSpec fileSpec;
    SInt16 resID;
};
typedef struct VoiceFileInfo VoiceFileInfo;
```

Fields

`fileSpec`
A file system specification structure that contains the volume, directory, and name of the file containing the voice. Generally, files containing a single voice are of type `kTextToSpeechVoiceFileType`, and files containing multiple voices are of type `kTextToSpeechVoiceBundleType`.

`resID`
The resource ID of the voice in the file. Voices are stored in resources of type `kTextToSpeechVoiceType`.

Discussion
A voice file information structure specifies the file in which a voice is stored and the resource ID of the voice within that file. Use the `GetVoiceInfo` (page 1641) function to obtain a voice file information structure for a voice.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechSynthesis.h

VoiceSpec

Defines a voice specification structure.

```
struct VoiceSpec {
    OSType creator;
    OSType id;
};
typedef struct VoiceSpec VoiceSpec;
typedef VoiceSpec * VoiceSpecPtr;
```

Fields

`creator`
The synthesizer that is required to use the voice. This is equivalent to the value contained in the `synthManufacturer` field of a speech version information structure and that contained in the `synthCreator` field of a speech extension data structure. The set of `OSType` values specified entirely by space characters and lowercase letters is reserved.

`id`
The voice ID of the voice for the synthesizer. Every voice on a synthesizer has a unique ID.

Discussion
A voice specification structure provides a unique specification that you must use to obtain information about a voice. You also must use a voice specification structure if you wish to create a speech channel that generates speech in a voice other than the current system default voice.

To ensure compatibility with future versions of the Speech Synthesis Manager, you should never fill in the fields of a voice specification structure yourself. Instead, you should create a voice specification structure by using the `MakeVoiceSpec` (page 1645) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechSynthesis.h`

Constants

Control Flags Constants

Flags that indicate which synthesizer features are active.

```
enum {
    kNoEndingProsody = 1,
    kNoSpeechInterrupt = 2,
    kPreflightThenPause = 4
};
```

Constants
`kNoEndingProsody`
Disables prosody at end of sentences. The `kNoEndingProsody` flag bit is used to control whether or not the speech synthesizer automatically applies ending prosody, the speech tone and cadence that normally occur at the end of a statement. Under normal circumstances (for example, when the flag bit is not set), ending prosody is applied to the speech when the end of the `textBuf` data is reached. This default behavior can be disabled by setting the `kNoEndingProsody` flag bit.

Some synthesizers do not speak until the `kNoEndingProsody` flag bit is reset, or they encounter a period in the text, or `textBuf` is full.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kNoSpeechInterrupt`

Does not interrupt current speech. The `kNoSpeechInterrupt` flag bit is used to control the behavior of `SpeakBuffer` when called on a speech channel that is still busy. When the flag bit is not set, `SpeakBuffer` behaves similarly to `SpeakString` and `SpeakText`. Any speech currently being produced on the specified speech channel is immediately interrupted, and then the new text buffer is spoken. When the `kNoSpeechInterrupt` flag bit is set, however, a request to speak on a channel that is still busy processing a prior text buffer will result in an error. The new buffer is ignored and the error `synthNotReady` is returned. If the prior text buffer has been fully processed, the new buffer is spoken normally. One way of achieving continuous speech without using callback functions is to continually call `SpeakBuffer` with the `kNoSpeechInterrupt` flag bit set until the function returns `noErr`. The function will then execute as soon as the first text buffer has been processed.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kPreflightThenPause`

Computes speech without generating.The `kPreflightThenPause` flag bit is used to minimize the latency experienced when the speech synthesizer is attempting to speak. Ordinarily, whenever a call to `SpeakString`, `SpeakText`, or `SpeakBuffer` is made, the speech synthesizer must perform a certain amount of initial processing before speech output is heard. This startup latency can vary from a few milliseconds to several seconds depending upon which speech synthesizer is being used. Recognizing that larger startup delays might be detrimental to certain applications, a mechanism exists to allow the synthesizer to perform any necessary computations at noncritical times. Once the computations have been completed, the speech is able to start instantly. When the `kPreflightThenPause` flag bit is set, the speech synthesizer will process the input text as necessary to the point where it is ready to begin producing speech output. At this point, the synthesizer will enter a paused state and return to the caller. When the application is ready to produce speech, it should call the `ContinueSpeech` function to begin speaking.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Discussion
These constants are used in the `controlFlags` parameter of the `SpeakBuffer` (page 1652) function and in the `synthFlagsl` field of the `SpeechVersionInfo` (page 1676) structure.

Declared In
`SpeechSynthesis.h`

Gender Constants

Constants that indicate the gender of the individual represented by a voice.

```
enum {
    kNeuter = 0,
    kMale = 1,
    kFemale = 2
};
```

Constants
`kNeuter`

Neuter voice.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kMale`
Male voice.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kFemale`
Female voice.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Discussion
These constants are used in the `gender` field of the `VoiceDescription` (page 1677) structure.

Declared In
`SpeechSynthesis.h`

Stop Speech Locations

Locations that indicate where speech should be paused or stopped.

```
enum {
    kImmediate = 0,
    kEndOfWord = 1,
    kEndOfSentence = 2
};
```

Constants
`kImmediate`
Speech should be paused or stopped immediately.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kEndOfWord`
Speech should be paused or stopped at the end of the word.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`kEndOfSentence`
Speech should be paused or stopped at the end of the sentence.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Discussion
See the functions `PauseSpeechAt` (page 1649) and `StopSpeechAt` (page 1658) for more information.

Declared In
`SpeechSynthesis.h`

Speech Synthesis Manager Operating System Types

The `OSType` definitions used by the Speech Synthesis Manager.

```
enum {
    kTextToSpeechSynthType = 'ttsc',
    kTextToSpeechVoiceType = 'ttvd',
    kTextToSpeechVoiceFileType = 'ttvf',
    kTextToSpeechVoiceBundleType = 'ttvb'
};
```

Constants

`kTextToSpeechSynthType`
The type of a synthesizer component.
Available in Mac OS X v10.0 and later.
Declared in `SpeechSynthesis.h`.

`kTextToSpeechVoiceType`
The type of a voice resource.
Available in Mac OS X v10.0 and later.
Declared in `SpeechSynthesis.h`.

`kTextToSpeechVoiceFileType`
The type of a voice file. Typically. files containing a single voice are of type `kTextToSpeechVoiceFileType`.
Available in Mac OS X v10.0 and later.
Declared in `SpeechSynthesis.h`.

`kTextToSpeechVoiceBundleType`
The type of a voice bundle file. Typically, files containing multiple voices are of type `kTextToSpeechVoiceBundleType`.
Available in Mac OS X v10.0 and later.
Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Speech-Channel Modes

The available text-processing and number-processing modes for a speech channel.

```
enum {
    modeText = 'TEXT',
    modePhonemes = 'PHON',
    modeNormal = 'NORM',
    modeLiteral = 'LTRL'
};
```

Constants

`modeText`
Indicates that the speech channel is in text-processing mode.
Available in Mac OS X v10.0 and later.
Declared in `SpeechSynthesis.h`.

`modePhonemes`
Indicates that the speech channel is in phoneme-processing mode. When in phoneme-processing mode, a text buffer is interpreted to be a series of characters representing various phonemes and prosodic controls.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`modeNormal`
Indicates that the synthesizer assembles digits into numbers (so that “12” is spoken as "twelve").

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`modeLiteral`
Indicates that each digit is spoken literally (so that “12” is spoken as "one, two").

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Speech-Channel Modes for Core Foundation-based Functions

The available text-processing and number-processing modes for a speech channel.

```
CFStringRef kSpeechModeText = CFSTR("TEXT");
CFStringRef kSpeechModePhoneme = CFSTR("PHON");
CFStringRef kSpeechModeNormal = CFSTR("NORM");
CFStringRef kSpeechModeLiteral = CFSTR("LTRL");
```

Constants
`kSpeechModeText`
Indicates that the speech channel is in text-processing mode.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechModePhoneme`
Indicates that the speech channel is in phoneme-processing mode. When in phoneme-processing mode, a text buffer is interpreted to be a series of characters representing various phonemes and prosodic controls.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechModeNormal`
Indicates that the synthesizer assembles digits into numbers (so that “12” is spoken as "twelve").

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechModeLiteral`
Indicates that each digit is spoken literally (so that “12” is spoken as "one, two").

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In
SpeechSynthesis.h

Voice Information Selectors

The types of voice data that can be requested by the GetVoiceInfo function.

```
enum {
    soVoiceDescription = 'info',
    soVoiceFile = 'fref'
};
```

Constants
soVoiceDescription
Get basic voice information.
Available in Mac OS X v10.0 and later.
Declared in SpeechSynthesis.h.

soVoiceFile
Get voice file reference information.
Available in Mac OS X v10.0 and later.
Declared in SpeechSynthesis.h.

Declared In
SpeechSynthesis.h

Speech-Channel Information Constants

Selectors that can be passed to the GetSpeechInfo or SetSpeechInfo functions.

```
enum {
    soStatus = 'stat',
    soErrors = 'erro',
    soInputMode = 'inpt',
    soCharacterMode = 'char',
    soNumberMode = 'nmbr',
    soRate = 'rate',
    soPitchBase = 'pbas',
    soPitchMod = 'pmod',
    soVolume = 'volm',
    soSynthType = 'vers',
    soRecentSync = 'sync',
    soPhonemeSymbols = 'phsy',
    soCurrentVoice = 'cvox',
    soCommandDelimiter = 'dlim',
    soReset = 'rset',
    soCurrentA5 = 'myA5',
    soRefCon = 'refc',
    soTextDoneCallBack = 'tdcb',
    soSpeechDoneCallBack = 'sdcb',
    soSyncCallBack = 'sycb',
    soErrorCallBack = 'ercb',
    soPhonemeCallBack = 'phcb',
    soWordCallBack = 'wdcb',
    soSynthExtension = 'xtnd',
    soSoundOutput = 'sndo',
    soOutputToFileWithCFURL = 'opaf'
};
```

Constants

`soStatus`

Get a speech status information structure for the speech channel. The `speechInfo` parameter is a pointer to a speech status information structure, described in `SpeechStatusInfo` (page 1674).

This selector works with the `GetSpeechInfo` (page 1638) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soErrors`

Get saved error information for the speech channel and clear its error registers. This selector lets you poll for various run-time errors that occur during speaking, such as the detection of badly formed embedded commands. Errors returned directly by Speech Synthesis Manager functions are not reported here. If your application defines an error callback function, the callback should use the `soErrors` selector to obtain error information. The `speechInfo` parameter is a pointer to a speech error information structure, described in `SpeechErrorInfo` (page 1673).

This selector works with the `GetSpeechInfo` (page 1638) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soInputMode`

Get or set the speech channel's current text-processing mode. The returned value specifies whether the channel is currently in text input mode or phoneme input mode. The `speechInfo` parameter is a pointer to a variable of type `OSType`, which specifies a text-processing mode. The constants `modeText` and `modePhonemes` specify the available text-processing modes.

The `modeText` constant indicates that the speech channel is in text-processing mode. The `modePhonemes` constant indicates that the speech channel is in phoneme-processing mode. When in phoneme-processing mode, a text buffer is interpreted to be a series of characters representing various phonemes and prosodic controls. Some synthesizers might support additional input-processing modes and define constants for these modes.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCharacterMode`

Get or set the speech channel's character-processing mode. Two constants are currently defined for the processing mode, `modeNormal` and `modeLiteral`. When the character-processing mode is `modeNormal`, input characters are spoken as you would expect to hear them. When the mode is `modeLiteral`, each character is spoken literally, so that the word "cat" would be spoken "C–A–T". The `speechInfo` parameter points to a variable of type `OSType`, which is the character-processing mode.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soNumberMode`

Get or set the speech channel's current number-processing mode. Two `OSType` constants are currently defined, `modeNormal` and `modeLiteral`. When the number-processing mode is `modeNormal`, the synthesizer assembles digits into numbers (so that 12 is spoken as "twelve"). When the mode is `modeLiteral`, each digit is spoken literally (so that 12 is spoken as "one, two"). The `speechInfo` parameter is a pointer to a variable of type `OSType`, which specifies the number-processing mode.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soRate`

Get or set a speech channel's speech rate. The `speechInfo` parameter is a pointer to a variable of type `Fixed`. The possible range of speech rates is from 0.000 to 65535.65535. The range of supported rates is not predefined by the Speech Synthesis Manager; each speech synthesizer provides its own range of speech rates. Average human speech occurs at a rate of 180 to 220 words per minute.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soPitchBase`

Get or set the speech channel's baseline speech pitch. This selector is intended for use by the Speech Synthesis Manager; ordinarily, an application uses the `GetSpeechPitch` (page 1639) and `SetSpeechPitch` (page 1651) functions. The `speechInfo` parameter is a pointer to a variable of type `Fixed`.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soPitchMod`

Get or set a speech channel's pitch modulation. The `speechInfo` parameter is a pointer to a variable of type `Fixed`. Pitch modulation is also expressed as a fixed-point value in the range of 0.000 to 127.000. These values correspond to MIDI note values, where 60.000 is equal to middle C on a piano scale. The most useful speech pitches fall in the range of 40.000 to 55.000. A pitch modulation value of 0.000 corresponds to a monotone in which all speech is generated at the frequency corresponding to the speech pitch. Given a speech pitch value of 46.000, a pitch modulation of 2.000 would mean that the widest possible range of pitches corresponding to the actual frequency of generated text would be 44.000 to 48.000.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soVolume`

Get or set the speech volume for a speech channel. The `speechInfo` parameter is a pointer to a variable of type `Fixed`. Volumes are expressed in fixed-point units ranging from 0.0 through 1.0. A value of 0.0 corresponds to silence, and a value of 1.0 corresponds to the maximum possible volume. Volume units lie on a scale that is linear with amplitude or voltage. A doubling of perceived loudness corresponds to a doubling of the volume.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSynthType`

Get a speech version information structure for the speech synthesizer being used on the specified speech channel. The `speechInfo` parameter is a pointer to a speech version information structure, described in `SpeechVersionInfo` (page 1676).

This selector works with the `GetSpeechInfo` (page 1638) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soRecentSync`

Get the message code for the most recently encountered synchronization command. If no synchronization command has been encountered, 0 is returned. The `speechInfo` parameter is a pointer to a variable of type `OSType`.

This selector works with the `GetSpeechInfo` (page 1638) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soPhonemeSymbols`

Get a list of phoneme symbols and example words defined for the speech channel's synthesizer. Your application might use this information to show the user what symbols to use when entering phonemic text directly. The `speechInfo` parameter is a pointer to a variable of type Handle that, on exit from the `GetSpeechInfo` function, is a handle to a phoneme descriptor structure, described in `PhonemeDescriptor` (page 1671).

This selector works with the `GetSpeechInfo` (page 1638) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCurrentVoice`

Set the current voice on the current speech channel to the specified voice. The `speechInfo` parameter is a pointer to a voice specification structure. Your application should create the structure by calling `MakeVoiceSpec` (page 1645). `SetSpeechInfo` will return an `incompatibleVoice` error if the specified voice is incompatible with the speech synthesizer associated with the speech channel. If you have a speech channel open using a voice from a particular synthesizer and you try to switch to a voice that works with a different synthesizer, you receive an `incompatibleVoice` error. You need to create a new channel to use with the new voice.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCommandDelimiter`

Set the embedded speech command delimiter characters to be used for the speech channel. By default the opening delimiter is "`[[`" and the closing delimiter is "`]]`". Your application might need to change these delimiters temporarily if those character sequences occur naturally in a text buffer that is to be spoken. Your application can also disable embedded command processing by passing empty delimiters (2 `NULL` bytes). The `speechInfo` parameter is a pointer to a delimiter information structure, described in `DelimiterInfo` (page 1670).

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soReset`

Set a speech channel back to its default state. For example, speech pitch and speech rate are set to default values. The `speechInfo` parameter should be set to `NULL`.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soCurrentA5`

Set the value that the Speech Synthesis Manager assigns to the A5 register before invoking any application-defined callback functions for the speech channel. The A5 register must be set correctly if the callback functions are to be able to access application global variables. The `speechInfo` parameter should be set to the pointer contained in the A5 register at a time when the application is not executing interrupt code or to `NULL` if your application wishes to clear a value previously set with the `soCurrentA5` selector.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soRefCon`

Set a speech channel's reference constant value. The reference constant value is passed to application-defined callback functions and might contain any value convenient for the application. The `speechInfo` parameter is a long integer containing the reference constant value. In contrast with other selectors, this selector does not require that the `speechInfo` parameter's value be a pointer value. Typically, however, an application does use this selector to pass a pointer or handle value to callback functions.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soTextDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished processing speech being generated on the speech channel. The `speechInfo` parameter is a pointer to an application-defined text-done callback function, whose syntax is described in `SpeechTextDoneProcPtr` (page 1667). Passing `NULL` in `speechInfo` disables the text-done callback function.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSpeechDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished generating speech on the speech channel. The `speechInfo` parameter is a pointer to an application-defined speech-done callback function, whose syntax is described in `SpeechDoneProcPtr` (page 1662). Passing `NULL` in `speechInfo` disables the speech-done callback function.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSyncCallBack`

Set the callback function to be called when the Speech Synthesis Manager encounters a synchronization command within an embedded speech command in text being processed on the speech channel. The `speechInfo` parameter is a pointer to an application-defined synchronization callback function, whose syntax is described in `SpeechSyncProcPtr` (page 1666). Passing `NULL` in `speechInfo` disables the synchronization callback function.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soErrorCallBack`

Set the callback function to be called when an error is encountered during the processing of an embedded command. The callback function might also be called if other conditions (such as insufficient memory) arise during the speech conversion process. When a Speech Synthesis Manager function returns an error directly, the error callback function is not called. The callback function is passed information about the most recent error; it can determine information about the oldest pending error by using the speech information selector `soErrors`. The `speechInfo` parameter is a pointer to an application-defined error callback function. Passing `NULL` in `speechInfo` disables the error callback function, `SpeechErrorProcPtr` (page 1664).

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soPhonemeCallBack`

Set the callback function to be called every time the Speech Synthesis Manager is about to generate a phoneme on the speech channel. The `speechInfo` parameter is a pointer to an application-defined phoneme callback function, whose syntax is described in `SpeechPhonemeProcPtr` (page 1665). Passing `NULL` in `speechInfo` disables the phoneme callback function.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soWordCallBack`

Set the callback function to be called every time the Speech Synthesis Manager is about to generate a word on the speech channel. The `speechInfo` parameter is a pointer to an application-defined word callback function, whose syntax is described in `SpeechWordProcPtr` (page 1669). Passing `NULL` in `speechInfo` disables the word callback function.

This selector works with the `SetSpeechInfo` (page 1650) function.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSynthExtension`

Get or set synthesizer-specific information or settings. The `speechInfo` parameter is a pointer to a speech extension data structure, described in `SpeechXtndData` (page 1677). Your application should set the `synthCreator` field of this structure before calling `GetSpeechInfo` (page 1638) or `SetSpeechInfo` (page 1650). Ordinarily, your application must pass additional information to the synthesizer in the `synthData` field.

This selector works with both the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soSoundOutput`

Get or set the speech channel's current output channel. (**Deprecated.** Use `soOutputToFileWithCFURL` (page 1692) instead.)

Available in Mac OS X v10.0 and later.

Declared in `SpeechSynthesis.h`.

`soOutputToFileWithCFURL`

Pass a `CFURLRef` to write to this file, `NULL` to generate sound.

This selector works with the `SetSpeechInfo` (page 1650) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

Discussion
See the `GetSpeechInfo` (page 1638) and `SetSpeechInfo` (page 1650) functions.

Declared In
`SpeechSynthesis.h`

Speech-Channel Properties

Properties used with `CopySpeechProperty` (page 1634) or `SetSpeechProperty` (page 1651) to get or set the characteristics of a speech channel.

```
CFStringRef kSpeechStatusProperty = CFSTR("stat");
CFStringRef kSpeechErrorsProperty = CFSTR("erro");
CFStringRef kSpeechInputModeProperty = CFSTR("inpt");
CFStringRef kSpeechCharacterModeProperty = CFSTR("char");
CFStringRef kSpeechNumberModeProperty = CFSTR("nmbr");
CFStringRef kSpeechRateProperty = CFSTR("rate");
CFStringRef kSpeechPitchBaseProperty = CFSTR("pbas");
CFStringRef kSpeechPitchModProperty = CFSTR("pmod");
CFStringRef kSpeechVolumeProperty = CFSTR("volm");
CFStringRef kSpeechSynthesizerInfoProperty = CFSTR("vers");
CFStringRef kSpeechRecentSyncProperty = CFSTR("sync");
CFStringRef kSpeechPhonemeSymbolsProperty = CFSTR("phsy");
CFStringRef kSpeechCurrentVoiceProperty = CFSTR("cvox");
CFStringRef kSpeechCommandDelimiterProperty = CFSTR("dlim");
CFStringRef kSpeechResetProperty = CFSTR("rset");
CFStringRef kSpeechOutputToFileURLProperty = CFSTR("opaf");
CFStringRef kSpeechRefConProperty = CFSTR("refc");
CFStringRef kSpeechTextDoneCallBack = CFSTR("tdcb");
CFStringRef kSpeechSpeechDoneCallBack = CFSTR("sdcb");
CFStringRef kSpeechSyncCallBack = CFSTR("sycb");
CFStringRef kSpeechPhonemeCallBack = CFSTR("phcb");
CFStringRef kSpeechErrorCFCallBack = CFSTR("eccb");
CFStringRef kSpeechWordCFCallBack = CFSTR("wccb");
```

Constants
`kSpeechStatusProperty`

Get speech-status information for the speech channel.

The value associated with this property is a `CFDictionary` object that contains speech-status information for the speech channel. See "`Speech Status Keys`" (page 1699) for a description of the keys present in the dictionary.

This property works with the `CopySpeechProperty` (page 1634) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechErrorsProperty`

Get speech-error information for the speech channel.

The value associated with this property is a `CFDictionary` object that contains speech-error information. See "Speech Error Keys" (page 1700) for a description of the keys present in the dictionary.

This property lets you get information about various run-time errors that occur during speaking, such as the detection of badly formed embedded commands. Errors returned directly by the Speech Synthesis Manager are not reported here. If your application defines an error callback function, the function can use this property to get error information.

This property works with the `CopySpeechProperty` (page 1634) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechInputModeProperty`

Get or set the speech channel's current text-processing mode.

The value associated with this property is a `CFString` object that specifies whether the channel is currently in text input mode or phoneme input mode. The constants `kSpeechModeText` and `kSpeechModePhoneme` (defined in "Speech-Channel Modes for Core Foundation-based Functions" (page 1684)) are the possible values of this string.

When in phoneme-processing mode, a text string is interpreted to be a series of characters representing various phonemes and prosodic controls. Some synthesizers might support additional input-processing modes and define constants for these modes.

This property works with the `CopySpeechProperty` (page 1634) and `SetSpeechProperty` (page 1651) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechCharacterModeProperty`

Get or set the speech channel's current character-processing mode.

The value associated with this property is a `CFString` object that specifies whether the speech channel is currently in normal or literal character-processing mode. The constants `kSpeechModeNormal` and `kSpeechModeLiteral` (defined in "Speech-Channel Modes for Core Foundation-based Functions" (page 1684)) are the possible values of this string.

When the character-processing mode is `kSpeechModeNormal`, input characters are spoken as you would expect to hear them. When the mode is `kSpeechModeLiteral`, each character is spoken literally, so that the word "cat" is spoken "C–A–T".

This property works with the `CopySpeechProperty` (page 1634) and `SetSpeechProperty` (page 1651) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechNumberModeProperty`

Get or set the speech channel's current number-processing mode.

The value associated with this property is a `CFString` object that specifies whether the speech channel is currently in normal or literal number-processing mode. The constants `kSpeechModeNormal` and `kSpeechModeLiteral` (defined in "Speech-Channel Modes for Core Foundation-based Functions" (page 1684)) are the possible values of this string.

When the number-processing mode is `kSpeechModeNormal`, the synthesizer assembles digits into numbers (so that "12" is spoken as "twelve"). When the mode is `kSpeechModeLiteral`, each digit is spoken literally (so that "12" is spoken as "one, two").

This property works with the `CopySpeechProperty` (page 1634) and `SetSpeechProperty` (page 1651) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechRateProperty`

Get or set a speech channel's speech rate.

The value associated with this property is a `CFNumber` object that specifies the speech channel's speaking rate.

The range of supported rates is not predefined by the Speech Synthesis Manager; each speech synthesizer provides its own range of speech rates. Average human speech occurs at a rate of 180 to 220 words per minute.

This property works with the `CopySpeechProperty` (page 1634) and `SetSpeechProperty` (page 1651) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPitchBaseProperty`

Get or set the speech channel's baseline speech pitch.

The value associated with this property is a `CFNumber` object that specifies the speech channel's baseline speech pitch.

Typical voice frequencies range from around 90 hertz for a low-pitched male voice to perhaps 300 hertz for a high-pitched child's voice. These frequencies correspond to approximate pitch values in the ranges of 30.000 to 40.000 and 55.000 to 65.000, respectively.

This property works with the `CopySpeechProperty` (page 1634) and `SetSpeechProperty` (page 1651) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPitchModProperty`

Get or set a speech channel's pitch modulation.

The value associated with this property is a `CFNumber` object that specifies the speech channel's pitch modulation.

Pitch modulation is also expressed as a floating-point value in the range of 0.000 to 127.000. These values correspond to MIDI note values, where 60.000 is equal to middle C on a piano scale. The most useful speech pitches fall in the range of 40.000 to 55.000. A pitch modulation value of 0.000 corresponds to a monotone in which all speech is generated at the frequency corresponding to the speech pitch. Given a speech pitch value of 46.000, a pitch modulation of 2.000 would mean that the widest possible range of pitches corresponding to the actual frequency of generated text would be 44.000 to 48.000.

This property works with the `CopySpeechProperty` (page 1634) and `SetSpeechProperty` (page 1651) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechVolumeProperty`

Get or set the speech volume for a speech channel.

The value associated with this property is a `CFNumber` object that specifies the speech channel's speech volume.

Volumes are expressed in floating-point values ranging from 0.0 through 1.0. A value of 0.0 corresponds to silence, and a value of 1.0 corresponds to the maximum possible volume. Volume units lie on a scale that is linear with amplitude or voltage. A doubling of perceived loudness corresponds to a doubling of the volume.

This property works with the `CopySpeechProperty` (page 1634) and `SetSpeechProperty` (page 1651) functions.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechSynthesizerInfoProperty`

Get information about the speech synthesizer being used on the specified speech channel.

The value associated with this property is a `CFDictionary` object that contains information about the speech synthesizer being used on the specified speech channel. See "`Speech Synthesizer Information Keys`" (page 1701) for a description of the keys present in the dictionary.

This property works with the `CopySpeechProperty` (page 1634) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechRecentSyncProperty`

Get the message code for the most recently encountered synchronization command.

The value associated with this property is a `CFNumber` object that specifies the most recently encountered synchronization command. This property works with the `CopySpeechProperty` (page 1634) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPhonemeSymbolsProperty`

Get a list of phoneme symbols and example words defined for the speech channel's synthesizer.

The value associated with this property is a `CFDictionary` object that contains the phoneme symbols and example words defined for the current synthesizer. Your application might use this information to show the user what symbols to use when entering phonemic text directly. See "Phoneme Symbols Keys" (page 1702) for a description of the keys present in the dictionary.

This property works with the `CopySpeechProperty` (page 1634) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechCurrentVoiceProperty`

Set the current voice on the current speech channel to the specified voice.

The value associated with this property is a `CFDictionary` object that contains the phoneme symbols and example words defined for the current synthesizer. Your application might use this information to show the user what symbols to use when entering phonemic text directly. See "Phoneme Symbols Keys" (page 1702) for the keys you can use to specify values in this dictionary.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechCommandDelimiterProperty`

Set the embedded speech command delimiter characters to be used for the speech channel.

By default, the opening delimiter is "`[[`" and the closing delimiter is "`]]`". Your application might need to change these delimiters temporarily if those character sequences occur naturally in a text buffer that is to be spoken. Your application can also disable embedded command processing by passing empty delimiters (as empty strings). The value associated with this property is a `CFDictionary` object that contains the delimiter information. See "Command Delimiter Keys" (page 1703) for the keys you can use to specify values in this dictionary.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechResetProperty`

Set a speech channel back to its default state.

You can use this function to, for example, set speech pitch and speech rate to default values. There is no value associated with this property; to reset the channel to its default state, set the string to `NULL`.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechOutputToFileURLProperty`

Set the speech output destination to a file or to the computer's speakers.

The value associated with this property is a `CFURL` object. To write the speech output to a file, use the file's `CFURLRef`; to generate the sound through the computer's speakers, use `NULL`.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechRefConProperty`

Set a speech channel's reference constant value.

The reference constant value is passed to application-defined callback functions and might contain any value convenient for the application. The value associated with this property is a `CFNumber` object that contains an integer value. For example, an application might set the value of the `CFNumber` object to an address in memory that contains a reference to an object or a pointer to a function.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechTextDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished processing speech being generated on the speech channel.

The value associated with this property is a `CFNumber` object whose value is a pointer to an application-defined text-done callback function, whose syntax is described in `SpeechTextDoneProcPtr` (page 1667). Passing a `CFNumber` object that contains the value `NULL` disables the text-done callback function.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechSpeechDoneCallBack`

Set the callback function to be called when the Speech Synthesis Manager has finished generating speech on the speech channel.

The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined speech-done callback function, whose syntax is described in `SpeechDoneProcPtr` (page 1662). Passing `NULL` for the value of this property disables the speech-done callback function.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechSyncCallBack`

Set the callback function to be called when the Speech Synthesis Manager encounters a synchronization command within an embedded speech command in text being processed on the speech channel.

The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined synchronization callback function, whose syntax is described in `SpeechSyncProcPtr` (page 1666). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the synchronization callback function.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPhonemeCallBack`

Set the callback function to be called every time the Speech Synthesis Manager is about to generate a phoneme on the speech channel.

The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined phoneme callback function, whose syntax is described in `SpeechPhonemeProcPtr` (page 1665). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the phoneme callback function.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechErrorCFCallBack`

Set the callback function to be called when an error is encountered during the processing of an embedded command.

When a Speech Synthesis Manager function returns an error directly, the error callback function is not called. The callback function is passed information about the most recent error; it can determine information about the oldest pending error by using the speech information property `kSpeechErrorsProperty`. The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined error callback function, whose syntax is described in `SpeechErrorCFProcPtr` (page 1663). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the error callback function.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechWordCFCallBack`

Set the callback function to be called every time the Speech Synthesis Manager is about to generate a word on the speech channel.

The value associated with this property is `CFNumber` object whose value is a pointer to an application-defined word callback function, whose syntax is described in `SpeechWordCFProcPtr` (page 1668). Passing a `CFNumber` object that contains the value `NULL` for the value of this property disables the word callback function.

This property works with the `SetSpeechProperty` (page 1651) function.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

Declared In
`SpeechSynthesis.h`

Synthesizer Option Keys

Keys used to specify synthesizer options.

```
CFStringRef kSpeechNoEndingProsody = CFSTR("NoEndingProsody");
CFStringRef kSpeechNoSpeechInterrupt = CFSTR("NoSpeechInterrupt");
CFStringRef kSpeechPreflightThenPause = CFSTR("PreflightThenPause");
```

Constants

`kSpeechNoEndingProsody`

Disable prosody at the end of sentences.

The `kSpeechNoEndingProsody` key is used to indicate whether the speech synthesizer should automatically apply ending prosody, which is the speech tone and cadence that normally occur at the end of a sentence. When the key is not specified, ending prosody is applied to the speech at the end of `aString`. This behavior can be disabled by specifying the `kSpeechNoEndingProsody` key in the `options` dictionary.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechNoSpeechInterrupt`

Do not interrupt current speech.

The `kSpeechNoSpeechInterrupt` key is used to control the behavior of `SpeakCFString` (page 1653) when it is called on a speech channel that is busy. When `kSpeechNoSpeechInterrupt` is not specified in the `options` dictionary, `SpeakCFString` immediately interrupts the speech currently being produced on the specified speech channel and the new `aString` text is spoken. When `kSpeechNoSpeechInterrupt` is specified in the `options` dictionary, the request to speak on a speech channel that is already busy causes the new `aString` text to be ignored and the `synthNotReady` error to be returned. As soon as the prior string has been fully processed, the new string is then spoken.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPreflightThenPause`

Compute speech without generating it.

The `kSpeechPreflightThenPause` key is used to minimize the latency experienced when the speech synthesizer is attempting to speak.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

Declared In

`SpeechSynthesis.h`

Speech Status Keys

Keys used with the `kSpeechStatusProperty` property to specify the status of the speech channel.

```
CFStringRef kSpeechStatusOutputBusy = CFSTR("OutputBusy");
CFStringRef kSpeechStatusOutputPaused = CFSTR("OutputPaused");
CFStringRef kSpeechStatusNumberOfCharactersLeft = CFSTR("NumberOfCharactersLeft");
CFStringRef kSpeechStatusPhonemeCode = CFSTR("PhonemeCode");
```

Constants

`kSpeechStatusOutputBusy`

Indicates whether the speech channel is currently producing speech.

A speech channel is considered to be producing speech even at some times when no audio data is being produced through the computer's speaker. This occurs, for example, when the Speech Synthesis Manager is processing input, but has not yet initiated speech or when speech output is paused.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechStatusOutputPaused`

Indicates whether speech output in the speech channel has been paused by a call to the `PauseSpeechAt` (page 1649) function.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechStatusNumberOfCharactersLeft`

The number of characters left in the input string of text.

When the value of this constant is zero, you can destroy the input string.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechStatusPhonemeCode`

The opcode for the phoneme that the speech channel is currently processing.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Speech Error Keys

Keys used with the `kSpeechErrorsProperty` property to describe errors encountered during speech processing and production.

```
CFStringRef kSpeechErrorCount = CFSTR("Count");
CFStringRef kSpeechErrorOldest = CFSTR("OldestCode");
CFStringRef kSpeechErrorOldestCharacterOffset = CFSTR("OldestCharacterOffset");
CFStringRef kSpeechErrorNewest = CFSTR("NewestCode");
CFStringRef kSpeechErrorNewestCharacterOffset = CFSTR("NewestCharacterOffset");
```

Constants

`kSpeechErrorCount`

The number of errors that have occurred in processing the current text string, since the last call to the `CopySpeechProperty` (page 1634) function with the `kSpeechErrorsProperty` property.

Using the `kSpeechErrorOldest` keys and the `kSpeechErrorNewest` keys, you can get information about the oldest and most recent errors that occurred since the last call to `CopySpeechProperty` (page 1634), but you cannot get information about any intervening errors.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechErrorOldest`

The error code of the first error that occurred since the last call to the `CopySpeechProperty` (page 1634) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechErrorOldestCharacterOffset`

The position in the text string of the first error that occurred since the last call to the `CopySpeechProperty` (page 1634) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechErrorNewest`

The error code of the most recent error that occurred since the last call to the `CopySpeechProperty` (page 1634) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechErrorNewestCharacterOffset`

The position in the text string of the most recent error that occurred since the last call to the `CopySpeechProperty` (page 1634) function with the `kSpeechErrorsProperty` property.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Speech Synthesizer Information Keys

Keys used with the `kSpeechSynthesizerInfoProperty` property to get information about the synthesizer.

```
CFStringRef kSpeechSynthesizerInfoIdentifier = CFSTR("Identifier");
CFStringRef kSpeechSynthesizerInfoVersion = CFSTR("Version");
```

Constants

`kSpeechSynthesizerInfoIdentifier`

The identifier of the speech synthesizer.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechSynthesizerInfoVersion`

The version of the speech synthesizer.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Phoneme Symbols Keys

Keys used with the `kSpeechPhonemeSymbolsProperty` property to provide information about the phoneme being processed.

```
CFStringRef kSpeechPhonemeInfoOpcode = CFSTR("Opcode");
CFStringRef kSpeechPhonemeInfoSymbol = CFSTR("Symbol");
CFStringRef kSpeechPhonemeInfoExample = CFSTR("Example");
CFStringRef kSpeechPhonemeInfoHiliteStart = CFSTR("HiliteStart");
CFStringRef kSpeechPhonemeInfoHiliteEnd = CFSTR("HiliteEnd");
```

Constants

`kSpeechPhonemeInfoOpcode`

The opcode of the phoneme.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechPhonemeInfoSymbol`

The symbol used to represent the phoneme.

The symbol does not necessarily have a phonetic connection to the phoneme, but might simply be an abstract textual representation of it.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechPhonemeInfoExample`

An example word that illustrates the use of the phoneme.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechPhonemeInfoHiliteStart`

The character offset into the example word that identifies the location of the beginning of the phoneme.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechPhonemeInfoHiliteEnd`
The character offset into the example word that identifies the location of the end of the phoneme.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Current Voice Keys

Keys used with the `kSpeechCurrentVoiceProperty` property to specify information about the current voice.

```
CFStringRef kSpeechVoiceCreator = CFSTR("Creator");
CFStringRef kSpeechVoiceID = CFSTR("ID");
```

Constants
`kSpeechVoiceCreator`
The synthesizer that is required to use the voice.

Declared in `SpeechSynthesis.h`.

Available in Mac OS X v10.5 and later.

`kSpeechVoiceID`
The voice ID of the voice for the synthesizer (every voice on a synthesizer has a unique ID).

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In
`SpeechSynthesis.h`

Command Delimiter Keys

Keys used with the `kSpeechCommandDelimiterProperty` property to specify information about the command delimiter strings.

```
CFStringRef kSpeechCommandPrefix = CFSTR("Prefix");
CFStringRef kSpeechCommandSuffix = CFSTR("Suffix");
```

Constants
`kSpeechCommandPrefix`
The command delimiter string that prefixes a command (by default, this is `[[`).

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

`kSpeechCommandSuffix`
The command delimiter string that suffixes a command (by default, this is `]]`).

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
SpeechSynthesis.h

Speech Dictionary Keys

Keys used in a speech dictionary to override the synthesizer's default pronunciation of a word.

```
CFStringRef kSpeechDictionaryLocaleIdentifier = CFSTR("LocaleIdentifier");
CFStringRef kSpeechDictionaryModificationDate = CFSTR("ModificationDate");
CFStringRef kSpeechDictionaryPronunciations = CFSTR("Pronunciations");
CFStringRef kSpeechDictionaryAbbreviations = CFSTR("Abbreviations");
CFStringRef kSpeechDictionaryEntrySpelling = CFSTR("Spelling");
CFStringRef kSpeechDictionaryEntryPhonemes = CFSTR("Phonemes");
```

Constants

`kSpeechDictionaryLocaleIdentifier`
The locale associated with the pronunciation.
Available in Mac OS X v10.5 and later.
Declared in SpeechSynthesis.h.

`kSpeechDictionaryModificationDate`
The date the dictionary was last modified.
Available in Mac OS X v10.5 and later.
Declared in SpeechSynthesis.h.

`kSpeechDictionaryPronunciations`
The set of custom pronunciations.
Available in Mac OS X v10.5 and later.
Declared in SpeechSynthesis.h.

`kSpeechDictionaryAbbreviations`
The set of custom pronunciations for abbreviations.
Available in Mac OS X v10.5 and later.
Declared in SpeechSynthesis.h.

`kSpeechDictionaryEntrySpelling`
The spelling of an entry.
Available in Mac OS X v10.5 and later.
Declared in SpeechSynthesis.h.

`kSpeechDictionaryEntryPhonemes`
The phonemic representation of an entry.
Available in Mac OS X v10.5 and later.
Declared in SpeechSynthesis.h.

Discussion
The keys in a speech dictionary can determine how a synthesizer pronounces a word. After you've created a speech dictionary, you register it with a speech channel with the `UseSpeechDictionary` (page 1661) function.

Declared In
SpeechSynthesis.h

Error Callback User-Information String

Specifies the string to speak to the user when an error occurs.

```
CFStringRef kSpeechErrorCallbackSpokenString = CFSTR("SpokenString");
```

Constants

`kSpeechErrorCallbackSpokenString`

The string to speak to the user when an error occurs.

Available in Mac OS X v10.5 and later.

Declared in `SpeechSynthesis.h`.

Declared In

`SpeechSynthesis.h`

Result Codes

The most common result codes returned by Speech Synthesis Manager are listed below.

Result Code	Value	Description
`noSynthFound`	-240	Could not find the specified speech synthesizer Available in Mac OS X v10.0 and later.
`synthOpenFailed`	-241	Could not open another speech synthesizer channel Available in Mac OS X v10.0 and later.
`synthNotReady`	-242	Speech synthesizer is still busy speaking Available in Mac OS X v10.0 and later.
`bufTooSmall`	-243	Output buffer is too small to hold result Available in Mac OS X v10.0 and later.
`voiceNotFound`	-244	Voice resource not found Available in Mac OS X v10.0 and later.
`incompatibleVoice`	-245	Specified voice cannot be used with synthesizer Available in Mac OS X v10.0 and later.
`badDictFormat`	-246	Pronunciation dictionary format error Available in Mac OS X v10.0 and later.
`badInputText`	-247	Raw phoneme text contains invalid characters Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Speech Synthesis Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Ticket Services Reference

Framework: ApplicationServices/ApplicationServices.h, Carbon/Carbon.h

Declared in PMTemplate.h
PMTicket.h

Overview

Ticket Services provides the functions and data types used to communicate printing information (page format and print settings) among the various modules in the printing system. Developers who write printing dialog extensions and printer modules need to use the functions described in this reference and also need to consult the *Printing Plug-in Interfaces Reference*. Developers who write applications do not need this reference; see the *Carbon Printing Manager Reference* instead.

Functions by Task

Managing Tickets

`PMTicketCreate` (page 1749)
Creates a new ticket.

`PMTicketValidate` (page 1795)
Validates a ticket against the constraint values specified in a job template.

`PMTicketRetain` (page 1774)
Increments the retention count of a ticket object.

`PMTicketRemoveTicket` (page 1773)
Removes a subticket from a ticket.

`PMTicketRelease` (page 1772)
Decrements the retention count of a ticket object.

`PMTicketReleaseAndClear` (page 1772)
Decrements the retention count of a ticket and sets the ticket reference to `NULL`.

`PMTicketCopy` (page 1747)
Copies a ticket.

`PMTicketCreateTemplate` (page 1749)
Retrieves a template from a ticket.

`PMTicketCopyItem` (page 1748)
Copies an item from one ticket to another ticket.

`PMTicketReleaseItem` (page 1773)
Removes an item from a ticket.

`PMTicketDeleteItem` (page 1750)
Makes an item in a ticket unavailable.

`PMTicketLockItem` (page 1770)
Locks an item in a ticket.

`PMTicketUnlockItem` (page 1794)
Unlocks an item in a ticket.

`PMTicketIsItemLocked` (page 1770)
Checks to see if an item in a ticket is locked.

`PMTicketConfirmTicket` (page 1746)
Checks whether a ticket appears to be valid.

`PMTicketContainsItem` (page 1746)
Checks whether an item exists in a ticket.

`PMTicketContainsTicket` (page 1747)
Checks whether a ticket is contained in another ticket.

`PMTicketFillFromArray` (page 1751)
Adds items defined in an array of ticket item structures to a ticket.

Setting Ticket Items

`PMTicketSetBoolean` (page 1774)
Writes an item of type `Boolean` to a ticket.

`PMTicketSetBytes` (page 1775)
Writes an item that's an array of type `UInt8` to a ticket.

`PMTicketSetCFArray` (page 1776)
Writes an item of type `CFArray` to a ticket.

`PMTicketSetCFBoolean` (page 1776)
Writes an item of type `CFBoolean` to a ticket.

`PMTicketSetCFData` (page 1777)
Writes an item of type `CFData` to a ticket.

`PMTicketSetCFDate` (page 1778)
Writes an item of type `CFDate` to a ticket.

`PMTicketSetCFDictionary` (page 1779)
Writes an item of type `CFDictionary` to a ticket.

`PMTicketSetCFNumber` (page 1779)
Writes an item of type `CFNumber` to a ticket.

`PMTicketSetCFString` (page 1780)
Writes an item of type `CFString` to a ticket.

`PMTicketSetCString` (page 1781)
Writes an item that's a C-style string to a ticket.

`PMTicketSetCStringArray` (page 1782)
Writes an item that's an array of C-style strings to a ticket.

PMTicketSetDouble (page 1782)
Writes an item of type double to a ticket.

PMTicketSetDoubleArray (page 1783)
Writes an item that's an array of values of type double to a ticket.

PMTicketSetItem (page 1784)
Writes an item of type CFType to a ticket.

PMTicketSetPMRect (page 1785)
Writes an item of type PMRect to a ticket.

PMTicketSetPMRectArray (page 1786)
Writes an item that's an array of values of type PMRect to a ticket.

PMTicketSetPMResolution (page 1787)
Writes an item of type PMResolution to a ticket.

PMTicketSetPMResolutionArray (page 1788)
Writes an item that's an array of data of type PMResolution to a ticket.

PMTicketSetPString (page 1789)
Writes an item that's a Pascal-style string to a ticket.

PMTicketSetSInt32 (page 1789)
Writes an item of type SInt32 to a ticket.

PMTicketSetSInt32Array (page 1790)
Writes an item that's an array of data of type SInt32 to a ticket.

PMTicketSetTemplate (page 1791)
Writes an item that's a job template to a ticket.

PMTicketSetTicket (page 1791)
Writes a subticket to a ticket.

PMTicketSetUInt32 (page 1792)
Writes an item of type UInt32 to a ticket.

PMTicketSetUInt32Array (page 1793)
Writes an item that's an array of data of type UInt32 to a ticket.

Getting Ticket Items

PMTicketGetAllocator (page 1751)
Obtains the allocator object used for allocating memory for a ticket.

PMTicketGetAPIVersion (page 1752)
Obtains the version of the application programming interface (API) used to created a ticket.

PMTicketGetBoolean (page 1752)
Obtains data for a ticket item of type Boolean.

PMTicketGetBytes (page 1753)
Obtains data for a ticket item that's an array of UInt8 values.

PMTicketGetCFArray (page 1754)
Obtains data for a ticket item that's a Core Foundation array.

PMTicketGetCFBoolean (page 1755)
Obtains data for a ticket item that's an array of CFBoolean values.

PMTicketGetCFData (page 1755)
Obtains data for a ticket item of type CFData.

PMTicketGetCFDate (page 1756)
Obtains data for a ticket item of type CFDate.

PMTicketGetCFDictionary (page 1757)
Obtains data for a ticket item of type CFDictionary.

PMTicketGetCFNumber (page 1758)
Obtains data for a ticket item of type CFNumber.

PMTicketGetCFString (page 1758)
Obtains data for a ticket item of type CFString.

PMTicketGetCString (page 1759)
Obtains data for a ticket item of that's a C-style string.

PMTicketGetDouble (page 1760)
Obtains data for a ticket item of type double.

PMTicketGetEnumType (page 1760)
Obtains a ticket's ticket type (job, paper, converter, and so on).

PMTicketGetIndexPMResolution (page 1761)
Obtains an indexed resolution for a ticket item that's an array of type PMResolution.

PMTicketGetItem (page 1762)
Obtains data for a ticket item of type CFType.

PMTicketGetLockedState (page 1762)
Obtains the lock state of a ticket.

PMTicketGetPMRect (page 1764)
Obtains data for a ticket item of type PMRect.

PMTicketGetPMResolution (page 1764)
Obtains data for a ticket item of type PMResolution.

PMTicketGetPPDDict (page 1765)
Obtains data for a ticket item that's a PostScript printer description (PPD) dictionary.

PMTicketGetPString (page 1766)
Obtains data for a ticket item that's a Pascal-style string.

PMTicketGetRetainCount (page 1767)
Obtains the retention count for a ticket.

PMTicketGetSInt32 (page 1767)
Obtains data for a ticket item of type SInt32.

PMTicketGetTicket (page 1768)
Obtains a subticket from a ticket.

PMTicketGetType (page 1769)
Obtains a value that specifies the ticket's type.

PMTicketGetUInt32 (page 1769)
Obtains an item of type UInt32 from a ticket.

Managing Templates

`PMTemplateCreate` (page 1714)
Creates a new job template.

`PMTemplateDelete` (page 1715)
Deletes an existing job template.

`PMTemplateMakeEntry` (page 1731)
Creates a new entry in a job template.

`PMTemplateMakeFullEntry` (page 1731)
Creates a new entry in a job template and sets the default and constraint values for the entry.

`PMTemplateMergeTemplates` (page 1732)
Merges the entries from one job template with entries from another job template.

`PMTemplateRemoveEntry` (page 1733)
Removes an entry from a job template.

`PMTemplateValidateItem` (page 1745)
Validates an item in a job template.

`PMTemplateIsLocked` (page 1729)
Checks to see if a job template is locked.

Setting Template Items

`PMTemplateSetBooleanDefaultValue` (page 1733)
Sets the default value for a job template entry of type `Boolean`.

`PMTemplateSetCFArrayConstraintValue` (page 1734)
Sets the constraint values for a job template entry of type `CFArray`.

`PMTemplateSetCFDataDefaultValue` (page 1734)
Sets the default value for a job template entry of type `CFDataRef`.

`PMTemplateSetCFDefaultValue` (page 1735)
Sets the default value for a job template entry of type `CFTypeRef`.

`PMTemplateSetCFRangeConstraint` (page 1735)
Sets a range of constraint values for a job template entry of type `CFTypeRef`.

`PMTemplateSetDoubleDefaultValue` (page 1736)
Sets the default value for a job template entry of type `double`.

`PMTemplateSetDoubleListConstraint` (page 1736)
Sets constraint values for a job template entry of type `double`.

`PMTemplateSetDoubleRangeDefaultValue` (page 1738)
Sets the default range of values for a job template entry of type `double`.

`PMTemplateSetDoubleRangeConstraint` (page 1737)
Sets a range of constraint values for a job template entry of type `double`.

`PMTemplateSetDoubleRangesConstraint` (page 1738)
Sets a range of constraint values for a job template entry that is a range of type `double`.

`PMTemplateSetPMRectDefaultValue` (page 1739)
Sets the default value for a job template entry of type `PMRect`.

PMTemplateSetPMRectListConstraint (page 1740)
Sets constraint values for a job template entry of type PMRect.

PMTemplateSetPMTicketDefaultValue (page 1741)
Sets the default value for a job template entry of type PMTicketRef.

PMTemplateSetPMTicketListConstraint (page 1741)
Sets constraint values for a job template entry of type PMTicketRef.

PMTemplateSetSInt32DefaultValue (page 1742)
Sets the default value for a job template entry of type SInt32.

PMTemplateSetSInt32ListConstraint (page 1742)
Sets constraint values for a job template entry of type SInt32.

PMTemplateSetSInt32RangeDefaultValue (page 1744)
Sets the default range of values for a job template entry of type SInt32.

PMTemplateSetSInt32RangeConstraint (page 1743)
Sets a range of constraint values for a job template entry of type SInt32.

PMTemplateSetSInt32RangesConstraint (page 1744)
Sets a range of constraint values for a job template entry that is a range of type SInt32.

Getting Template Items

PMTemplateGetValueType (page 1729)
Obtains the data type of a job template entry.

PMTemplateGetConstraintType (page 1718)
Obtains the constraint type for a job template entry.

PMTemplateGetBooleanDefaultValue (page 1715)
Obtains the default value for a job template entry of type Boolean.

PMTemplateGetCFArrayConstraintValue (page 1716)
Obtains an array of constraint values for a job template entry of type CFArrayRef.

PMTemplateGetCFDataDefaultValue (page 1716)
Obtains the default value for a job template entry of type CFDataRef.

PMTemplateGetCFDefaultValue (page 1717)
Obtains the default value for a job template entry of type CFTypeRef.

PMTemplateGetCFRangeConstraintValue (page 1717)
Obtains the range of constraint values for a job template entry of type CFTypeRef.

PMTemplateGetDoubleDefaultValue (page 1719)
Obtains the default value for a job template entry of type double.

PMTemplateGetDoubleRangeDefaultValue (page 1721)
Obtains the default range values for a job template entry of type double.

PMTemplateGetDoubleListConstraintValue (page 1719)
Obtains constraint values for a job template entry of type double.

PMTemplateGetDoubleRangeConstraintValue (page 1720)
Obtains the default range of values for a job template entry of type double.

PMTemplateGetDoubleRangesConstraintValue (page 1722)
Obtains a range for the minimum and maximum constraint values for a job template entry that is a range of type double.

PMTemplateGetPMRectDefaultValue (page 1723)
Obtains the default value for a job template entry of type PMRect.

PMTemplateGetPMRectListConstraintValue (page 1724)
Obtains constraint values for a job template entry of type PMRect.

PMTemplateGetPMTicketDefaultValue (page 1724)
Obtains the default value for a job template entry of type PMTicket.

PMTemplateGetListTicketConstraintValue (page 1722)
Obtains the constraint value for a job template entry that of type PMTicketRef.

PMTemplateGetSInt32DefaultValue (page 1725)
Obtains the default value for a job template entry of type SInt32.

PMTemplateGetSInt32RangeDefaultValue (page 1727)
Obtains the default range values for a job template entry of type SInt32.

PMTemplateGetSInt32ListConstraintValue (page 1726)
Obtains constraint values for a job template entry of type SInt32.

PMTemplateGetSInt32RangeConstraintValue (page 1726)
Obtains the default range of values for a job template entry of type SInt32.

PMTemplateGetSInt32RangesConstraintValue (page 1728)
Obtains a range of constraint values for a job template entry that is a range of type SInt32.

Converting To and From XML

PMTemplateCreateXML (page 1714)
Converts the data in a job template to data represented as XML.

PMTicketToXML (page 1794)
Converts the data in a ticket to XML.

PMTicketWriteXML (page 1795)
Converts a ticket to XML and then writes it to a file stream.

PMTicketWriteXMLToFile (page 1796)
Converts a ticket to XML and then writes it to a file.

PMXMLToTicket (page 1796)
Converts a ticket saved as XML to a ticket.

PMTemplateLoadFromXML (page 1730)
Restores a job template that was previously converted to XML.

PMTicketReadXMLFromFile (page 1771)
Restores a ticket previously converted to XML and saved as a file.

Deprecated Functions

PMTicketSetMetaItem (page 1785)
Writes an item that does not need to be stored in an XML-representation of a ticket.

PMTicketGetMetaItem (page 1763)
Obtains data for a item added with the function PMTicketSetMetaItem.

Functions

PMTemplateCreate

Creates a new job template.

```
OSStatus PMTemplateCreate (
   PMTemplateRef *newTemplate
);
```

Parameters

newTemplate

On return, points to a job template data structure allocated by the function.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateCreateXML

Converts the data in a job template to data represented as XML.

```
OSStatus PMTemplateCreateXML (
   PMTemplateRef srcTemplate,
   CFDataRef *xmlData
);
```

Parameters

srcTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

xmlData

On return, points to a reference to an XML representation of the data in the job template specified by the `srcTemplate` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateDelete

Deletes an existing job template.

```
OSStatus PMTemplateDelete (
   PMTemplateRef *oldTemplate
);
```

Parameters

oldTemplate

On input, a reference to a job template created by calling the function `PMTemplateCreate`. On return, points to `NULL`.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateGetBooleanDefaultValue

Obtains the default value for a job template entry of type `Boolean`.

```
OSStatus PMTemplateGetBooleanDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   Boolean *defaultValue
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

defaultValue

On return, points to the default value for the template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateGetCFArrayConstraintValue

Obtains an array of constraint values for a job template entry of type `CFArrayRef`.

```
OSStatus PMTemplateGetCFArrayConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFArrayRef *constraintValue
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

constraintValue

On return, a poinrter to a Core Foundation array reference for the constraint values for the template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateGetCFDataDefaultValue

Obtains the default value for a job template entry of type `CFDataRef`.

```
OSStatus PMTemplateGetCFDataDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFDataRef *defaultValue
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

defaultValue

On return, points to a Core Foundation data reference that specfies the default data for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetCFDefaultValue

Obtains the default value for a job template entry of type `CFTypeRef`.

```
OSStatus PMTemplateGetCFDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFTypeRef *defaultValue
);
```

Parameters

pmTemplate
A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key
A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

defaultValue
On return, points to a Core Foundation type reference that specifies the default data for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetCFRangeConstraintValue

Obtains the range of constraint values for a job template entry of type `CFTypeRef`.

```
OSStatus PMTemplateGetCFRangeConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFTypeRef *min,
   CFTypeRef *max
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

min

On return, points to a Core Foundation type reference that specifies the minimum value to which the template entry specified by the `key` parameter is constrained.

max

On return, points to a Core Foundation type reference that specifies the maximum value to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetConstraintType

Obtains the constraint type for a job template entry.

```
OSStatus PMTemplateGetConstraintType (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMConstraintType *constraintType
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint type you want to retrieve from the template.

constraintType

On return, points to the constraint type for the data specified by the `key` parameter. See "Constraint Types" (page 1802) for a list of the constraint types that can be returned.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetDoubleDefaultValue

Obtains the default value for a job template entry of type `double`.

```
OSStatus PMTemplateGetDoubleDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double *defaultValue
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

defaultValue

On return, points to the default `double` value for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetDoubleListConstraintValue

Obtains constraint values for a job template entry of type `double`.

```
OSStatus PMTemplateGetDoubleListConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   int *listSize,
   double *doubleList
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

listSize

A pointer to the number of items in the array specified by the parameter `doubleList`.

doubleList

On return, points to an array of `double` values to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
You need to call the function `PMTemplateGetDoubleListConstraintValue` twice. First, call the function to get the number of items in the array specified by the parameter `doubleList`. You should call the function again after you allocate space for the array.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetDoubleRangeConstraintValue

Obtains the default range of values for a job template entry of type `double`.

```
OSStatus PMTemplateGetDoubleRangeConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double *min,
   double *max
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

min

On return, points to the minimum value to which the template entry specified by the `key` parameter is constrained.

max

On return, points to the maximum value to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetDoubleRangeDefaultValue

Obtains the default range values for a job template entry of type `double`.

```
OSStatus PMTemplateGetDoubleRangeDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double *min,
   double *max
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

min

On return, points to the minimum value to which the range is constrained for the template entry specified by the `key` parameter.

max

On return, points to the maximum value to which the range is constrained for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTemplate.h

PMTemplateGetDoubleRangesConstraintValue

Obtains a range for the minimum and maximum constraint values for a job template entry that is a range of type `double`.

```
OSStatus PMTemplateGetDoubleRangesConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double *minForMin,
   double *maxForMin,
   double *minForMax,
   double *maxForMax
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

minForMin

On return, points to the minimum value to which the minimum of the range is constrained for the template entry specified by the `key` parameter.

maxForMin

On return, points to the maximum value to which the minimum of the range is constrained for the template entry specified by the `key` parameter.

minForMax

On return, points to the minimum value to which the maximum of the range is constrained for the template entry specified by the `key` parameter.

maxForMax

On return, points to the maximum value to which the maximum of the range is constrained for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTemplate.h

PMTemplateGetListTicketConstraintValue

Obtains the constraint value for a job template entry that of type `PMTicketRef`.

```
OSStatus PMTemplateGetListTicketConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMTicketRef *listTicket
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

listTicket

On return, points to a reference to the list ticket to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetPMRectDefaultValue

Obtains the default value for a job template entry of type `PMRect`.

```
OSStatus PMTemplateGetPMRectDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMRect *defaultValue
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

defaultValue

On return, points to the default `PMRect` data for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetPMRectListConstraintValue

Obtains constraint values for a job template entry of type `PMRect`.

```
OSStatus PMTemplateGetPMRectListConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   int *listSize,
   PMRect *rectList
);
```

Parameters

pmTemplate
A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key
A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

listSize
A pointer to the number of items in the array specified by the parameter `rectList`.

rectList
On return, points to an array of `PMRect` values to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
You need to call the function `PMTemplateGetPMRectListConstraintValue` twice. First, call the function to get the number of items in the array specified by the parameter `rectList`. You should call the function again after you allocate space for the array.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetPMTicketDefaultValue

Obtains the default value for a job template entry of type `PMTicket`.

```
OSStatus PMTemplateGetPMTicketDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMTicketRef *defaultValue
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

defaultValue

On return, points to a reference to the default ticket for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetSInt32DefaultValue

Obtains the default value for a job template entry of type `SInt32`.

```
OSStatus PMTemplateGetSInt32DefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 *defaultValue
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

defaultValue

On return, points to the default value for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetSInt32ListConstraintValue

Obtains constraint values for a job template entry of type `SInt32`.

```
OSStatus PMTemplateGetSInt32ListConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   int *listSize,
   SInt32 *sint32List
);
```

Parameters

pmTemplate
A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key
A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

listSize
On return, points to the number of items in the array specified by the parameter `sint32List`.

sint32List
On return, points to an array of `SInt32` values to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
You need to call the function `PMTemplateGetSInt32ListConstraintValue` twice. First, call the function to get the number of items in the array specified by the parameter `sint32List`. You should call the function again after you allocate space for the array.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetSInt32RangeConstraintValue

Obtains the default range of values for a job template entry of type `SInt32`.

```
OSStatus PMTemplateGetSInt32RangeConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 *min,
   SInt32 *max
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

min

On return, points to the minimum value to which the template entry specified by the `key` parameter is constrained.

max

On return, points to the maximum value to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetSInt32RangeDefaultValue

Obtains the default range values for a job template entry of type `SInt32`.

```
OSStatus PMTemplateGetSInt32RangeDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 *min,
   SInt32 *max
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

min

On return, points to the minimum value to which the range is constrained for the template entry specified by the `key` parameter.

max

On return, points to the maximum value to which the range is constrained for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetSInt32RangesConstraintValue

Obtains a range of constraint values for a job template entry that is a range of type `SInt32`.

```
OSStatus PMTemplateGetSInt32RangesConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 *minForMin,
   SInt32 *maxForMin,
   SInt32 *minForMax,
   SInt32 *maxForMax
);
```

Parameters

pmTemplate

A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key

A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

minForMin

On return, points to the minimum value to which the minimum of the range is constrained for the template entry specified by the `key` parameter.

maxForMin

On return, points to the maximum value to which the minimum of the range is constrained for the template entry specified by the `key` parameter.

minForMax

On return, points to the minimum value to which the maximum of the range is constrained for the template entry specified by the `key` parameter.

maxForMax

On return, points to the maximum value to which the maximum of the range is constrained for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateGetValueType

Obtains the data type of a job template entry.

```
OSStatus PMTemplateGetValueType (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMValueType *valueType
);
```

Parameters

pmTemplate
A reference to a job template. You can call the function `PMSessionGetDataFromSession` with the ticket type set to `kPDE_PMJobTemplateRef` to get a value that you can then cast to a job template reference.

key
A reference to a `CFString` object that uniquely identifies the data you want to retrieve from the template.

valueType
On return, points to the value type for the template entry specified by the `key` parameter. See "PMTicketItemType" (page 1800) for a list of values that can be returned.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateIsLocked

Checks to see if a job template is locked.

```
OSStatus PMTemplateIsLocked (
   PMTemplateRef srcTemplate,
   Boolean *locked
);
```

Parameters

srcTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

locked

On return, points to `true` if the job template is locked and `false` if the template is not locked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
A job template cannot be modified unless it is unlocked.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateLoadFromXML

Restores a job template that was previously converted to XML.

```
OSStatus PMTemplateLoadFromXML (
   CFDataRef srcData,
   PMTemplateRef *destTemplate
);
```

Parameters

srcData

A reference to `CFData` that represents job template data. You obtain a `CFDataRef` by calling the function `PMTemplateCreateXML`.

destTemplate

A reference to a job template created by calling the function `PMTemplateCreate`. On return, the job template contains the entries specified by the XML data.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateMakeEntry

Creates a new entry in a job template.

```
OSStatus PMTemplateMakeEntry (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMValueType valueType,
   PMConstraintType constraintType
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the entry. You use the key to set and obtain the data for this entry.

valueType

The data type of the value associated with the entry. See "PMValueType" (page 1801) for a list of constants you can use to specify the data type.

constraintType

The type of constraint you want applied to the data associated with the entry. See "Constraint Types" (page 1802) for a list of constants you can use to specify the constraint type.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Discussion

After you call the function `PMTemplateMakeEntry`, you should set default and constraint values for the entry by calling the appropriate `PMTemplateSet` functions for the data type of the entry.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateMakeFullEntry

Creates a new entry in a job template and sets the default and constraint values for the entry.

```
OSStatus PMTemplateMakeFullEntry (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMValueType valueType,
   PMConstraintType constraintType,
   CFTypeRef defaultValue,
   CFTypeRef constraintValue
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the entry. You use the key to set and obtain the data for this entry.

valueType

The data type of the value associated with the entry. See "PMValueType" (page 1801) for a list of constants you can use to specify the data type.

constraintType

The type of constraint you want applied to the data associated with the entry. See "Constraint Types" (page 1802) for a list of constants you can use to specify the constraint type.

defaultValue

A reference to the default value for this entry.

constraintValue

A reference to the values used to constrain the data associated with this entry.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateMergeTemplates

Merges the entries from one job template with entries from another job template.

```
OSStatus PMTemplateMergeTemplates (
   PMTemplateRef sourceTemplate,
   PMTemplateRef destTemplate
);
```

Parameters

sourceTemplate

A reference to the job template whose entries you want to merge with the destination template.

destTemplate

A reference to the job template into which you want to merge entries from the source template. Any entry in the destination template that has the same key as an entry in the source template is overwritten by the data from the source template.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateRemoveEntry

Removes an entry from a job template.

```
OSStatus PMTemplateRemoveEntry (
   PMTemplateRef pmTemplate,
   CFStringRef key
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the entry you want to remove.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateSetBooleanDefaultValue

Sets the default value for a job template entry of type `Boolean`.

```
OSStatus PMTemplateSetBooleanDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   Boolean defaultValue
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose default value you want to set.

defaultValue

A `Boolean` value that specifies the default value for template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateSetCFArrayConstraintValue

Sets the constraint values for a job template entry of type `CFArray`.

```
OSStatus PMTemplateSetCFArrayConstraintValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFArrayRef constraintValue
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint values you want to set.

constraintValue

A reference to a Core Foundation array that contains the values to which the template entry specified by the `key` parameter is constrained.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateSetCFDataDefaultValue

Sets the default value for a job template entry of type `CFDataRef`.

```
OSStatus PMTemplateSetCFDataDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFDataRef defaultValue
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose default value you want to set.

defaultValue

A reference to the default data for template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In
PMTemplate.h

PMTemplateSetCFDefaultValue

Sets the default value for a job template entry of type CFTypeRef.

```
OSStatus PMTemplateSetCFDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFTypeRef defaultValue
);
```

Parameters

pmTemplate
A reference to a job template created by calling the function PMTemplateCreate.

key
A reference to a CFString object that uniquely identifies the data whose default value you want to set.

defaultValue
A reference to the default data for template entry specified by the key parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTemplate.h

PMTemplateSetCFRangeConstraint

Sets a range of constraint values for a job template entry of type CFTypeRef.

```
OSStatus PMTemplateSetCFRangeConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFTypeRef min,
   CFTypeRef max
);
```

Parameters

pmTemplate
A reference to a job template created by calling the function PMTemplateCreate.

key
A reference to a CFString object that uniquely identifies the data whose range of constraint values you want to set.

min

A reference to the minimum value to which the template entry specified by the `key` parameter is constrained.

max

A reference to the maximum value to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetDoubleDefaultValue

Sets the default value for a job template entry of type `double`.

```
OSStatus PMTemplateSetDoubleDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double defaultValue
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose default value you want to set.

defaultValue

A `double` value that specifies the default value for template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetDoubleListConstraint

Sets constraint values for a job template entry of type `double`.

```
OSStatus PMTemplateSetDoubleListConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   int listSize,
   double *doubleList
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint values you want to set.

listSize

The size of the array specified by the parameter `doubleList`.

doubleList

A pointer to an array of `double` values to which the template entry specified by the `key` parameter is constrained.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateSetDoubleRangeConstraint

Sets a range of constraint values for a job template entry of type `double`.

```
OSStatus PMTemplateSetDoubleRangeConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double min,
   double max
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint values you want to set.

min

A `double` value that specifies the minimum value to which the template entry specified by the `key` parameter is constrained.

max

A `double` value that specifies the maximum value to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetDoubleRangeDefaultValue

Sets the default range of values for a job template entry of type `double`.

```
OSStatus PMTemplateSetDoubleRangeDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double min,
   double max
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose default values you want to set.

min

A `double` value that specifies the default minimum value for the template entry specified by the `key` parameter.

max

A `double` value that specifies the default maximum value for template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetDoubleRangesConstraint

Sets a range of constraint values for a job template entry that is a range of type `double`.

```
OSStatus PMTemplateSetDoubleRangesConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   double minForMin,
   double maxForMin,
   double minForMax,
   double maxForMax
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose range of constraint values you want to set.

minForMin

A `double` value that specifies the minimum value to which the minimum of the range is constrained for the template entry specified by the `key` parameter.

maxForMin

A `double` value that specifies the maximum value to which the minimum of the range is constrained for the template entry specified by the `key` parameter.

minForMax

A `double` value that specifies the minimum value to which the maximum of the range is constrained for the template entry specified by the `key` parameter.

maxForMax

A `double` value that specifies the maximum value to which the maximum of the range is constrained for the template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateSetPMRectDefaultValue

Sets the default value for a job template entry of type `PMRect`.

```
OSStatus PMTemplateSetPMRectDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMRect *defaultValue
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose default value you want to set.

defaultValue

A pointer to a `PMRect` that specifies the default rectangle for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetPMRectListConstraint

Sets constraint values for a job template entry of type `PMRect`.

```
OSStatus PMTemplateSetPMRectListConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   int listSize,
   PMRect *rectList
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint values you want to set.

listSize

The size of the array specified by the parameter `rectList`.

rectList

A pointer to an array of `PMRect` values to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetPMTicketDefaultValue

Sets the default value for a job template entry of type `PMTicketRef`.

```
OSStatus PMTemplateSetPMTicketDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMTicketRef defaultValue
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose default value you want to set.

defaultValue

A reference to a ticket that specifies the default ticket for the template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateSetPMTicketListConstraint

Sets constraint values for a job template entry of type `PMTicketRef`.

```
OSStatus PMTemplateSetPMTicketListConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   PMTicketRef listTicket
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint values you want to set.

listTicket

A reference to a ticket that specifies the default list ticket for the template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetSInt32DefaultValue

Sets the default value for a job template entry of type `SInt32`.

```
OSStatus PMTemplateSetSInt32DefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 defaultValue
);
```

Parameters

`pmTemplate`
A reference to a job template created by calling the function `PMTemplateCreate`.

`key`
A reference to a `CFString` object that uniquely identifies the data whose default value you want to set.

`defaultValue`
An `SInt32` value that specifies the default value for the template entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetSInt32ListConstraint

Sets constraint values for a job template entry of type `SInt32`.

```
OSStatus PMTemplateSetSInt32ListConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   int listSize,
   SInt32 *sint32List
);
```

Parameters

`pmTemplate`
A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint values you want to set.

listSize

The size of the array specified by the parameter `sint32List`.

sint32List

A pointer to an array of `SInt32` values to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetSInt32RangeConstraint

Sets a range of constraint values for a job template entry of type `SInt32`.

```
OSStatus PMTemplateSetSInt32RangeConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 min,
   SInt32 max
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose constraint values you want to set.

min

An `SInt32` value that specifies the minimum value to which the template entry specified by the `key` parameter is constrained.

max

An `SInt32` value that specifies the maximum value to which the template entry specified by the `key` parameter is constrained.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTemplate.h`

PMTemplateSetSInt32RangeDefaultValue

Sets the default range of values for a job template entry of type `SInt32`.

```
OSStatus PMTemplateSetSInt32RangeDefaultValue (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 min,
   SInt32 max
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose default values you want to set.

min

An `SInt32` value that specifies the default minimum value for the template entry specified by the `key` parameter.

max

An `SInt32` value that specifies the default maximum value for the template entry specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTemplate.h`

PMTemplateSetSInt32RangesConstraint

Sets a range of constraint values for a job template entry that is a range of type `SInt32`.

```
OSStatus PMTemplateSetSInt32RangesConstraint (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   SInt32 minForMin,
   SInt32 maxForMin,
   SInt32 minForMax,
   SInt32 maxForMax
);
```

Parameters

pmTemplate

A reference to a job template created by calling the function `PMTemplateCreate`.

key

A reference to a `CFString` object that uniquely identifies the data whose range of constraint values you want to set.

minForMin
An SInt32 value that specifies the minimum value to which the minimum of the range is constrained for the template entry specified by the key parameter.

maxForMin
An SInt32 value that specifies the maximum value to which the minimum of the range is constrained for the template entry specified by the key parameter.

minForMax
An SInt32 value that specifies the minimum value to which the maximum of the range is constrained for the template entry specified by the key parameter.

maxForMax
An SInt32 value that specifies the maximum value to which the maximum of the range is constrained for the template entry specified by the key parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTemplate.h

PMTemplateValidateItem

Validates an item in a job template.

```
OSStatus PMTemplateValidateItem (
   PMTemplateRef pmTemplate,
   CFStringRef key,
   CFTypeRef item,
   Boolean *validationResults
);
```

Parameters

pmTemplate
A reference to a job template created by calling the function PMTemplateCreate.

key
A reference to a CFString object that uniquely identifies the job template entry you want to validate.

item
A reference to the data you want to validate.

validationResults
On return, points to true is the item is validated and false if it is not validated.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTemplate.h

PMTicketConfirmTicket

Checks whether a ticket appears to be valid.

```
OSStatus PMTicketConfirmTicket (
   PMTicketRef ticket
);
```

Parameters

ticket
A reference to a ticket created by calling the function PMTicketCreate.

Return Value
A result code. Returns noErr if the ticket appears to be valid. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketContainsItem

Checks whether an item exists in a ticket.

```
Boolean PMTicketContainsItem (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key
);
```

Parameters

ticket
A reference to a ticket created by calling the function PMTicketCreate.

nodeIndex1
Reserved for future use. Currently, you must pass the constant kPMTopLevel.

nodeIndex2
Reserved for future use. Currently, you must pass the constant kPMTopLevel.

key
A reference to a CFString object that uniquely identifies the ticket item you want to check.

Return Value
Returns true if the item exists.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketContainsTicket

Checks whether a ticket is contained in another ticket.

```
OSStatus PMTicketContainsTicket (
   PMTicketRef ticket,
   CFStringRef requestedType,
   UInt32 index,
   Boolean *exists
);
```

Parameters

ticket
A reference to a ticket created by calling the function PMTicketCreate.

requestedType
A reference to a CFString object that specfies the ticket type of the ticket for which you want to check. See "Ticket Type Strings" (page 1831) for a list of constants you can pass.

index
Reserved for future use. Currently, you must pass the constant kPMTopLevel.

exists
On return, points to true if the ticket contains a subticket of the ticket type specified by the requestedType parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketCopy

Copies a ticket.

```
OSStatus PMTicketCopy (
   CFAllocatorRef allocator,
   PMTicketRef sourceTicket,
   PMTicketRef *destinationTicket
);
```

Parameters

allocator
A reference to the allocator object to be used for allocating memory. Pass a reference to a valid allocator or kCFAllocatorDefault to request the default allocator.

sourceTicket

A reference to a ticket created by calling the function `PMTicketCreate`. This is the ticket you want to copy.

destinationTicket

A pointer to a ticket reference. On return, points to a copy of the source ticket.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketCopyItem

Copies an item from one ticket to another ticket.

```
OSStatus PMTicketCopyItem (
   PMTicketRef sourceTicket,
   PMTicketRef destTicket,
   CFStringRef clientID,
   CFStringRef key,
   Boolean locked
);
```

Parameters

sourceTicket

A reference to a ticket created by calling the function `PMTicketCreate`. This is the ticket from which you want to copy an item.

destTicket

A reference to a ticket created by calling the function `PMTicketCreate`. This is the ticket to which the item is copied.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to copy.

locked

A `Boolean` value. Pass `true` to lock the copied item or `false` to set the item's lock state to unlock. Locking the item prevents any subsequent modification of this item.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
If an item must be copied from one ticket to another, call the function `PMTicketCopyItem` to make the simple transfer. This updates the modification date and client ID for the item. The *locked* field determines if subsequent updates can be made.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketCreate

Creates a new ticket.

```
OSStatus PMTicketCreate (
   CFAllocatorRef allocator,
   CFStringRef ticketType,
   PMTicketRef *newTicket
);
```

Parameters

allocator

A reference to the allocator object to be used for allocating memory. Pass a reference to a valid allocator or `kCFAllocatorDefault` to request the default allocator.

ticketType

A reference to a `CFString` object that specfies the ticket type of the ticket you want to create. See "Ticket Type Strings" (page 1831) for a list of constants you can pass.

newTicket

On return, a reference to a ticket.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketCreateTemplate

Retrieves a template from a ticket.

```
OSStatus PMTicketCreateTemplate (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   PMTemplateRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key
A reference to a `CFString` object that uniquely identifies the template you want to retrieve.

item
On return, points to the template reference specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
The function `PMTicketCreateTemplate` retrieves the template data associated with the specified key. The template is stored in the ticket as flattened data. The function creates a template object from the flattened data.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketDeleteItem

Makes an item in a ticket unavailable.

```
OSStatus PMTicketDeleteItem (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to make unavailable.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
After an item is deleted by calling the function `PMTicketDeleteItem`, the result code `kPMInvalidItem` is returned when anyone tries to access the item. The function `PMTicketDeleteItem` actually makes the item unavailable rather than deleting the item. The function adds information to the item's dictionary to record the history of the item and to indicate the item is unavailable. You should call the function `PMTicketReleaseItem` if you want to completely remove the item from the ticket.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketFillFromArray

Adds items defined in an array of ticket item structures to a ticket.

```
OSStatus PMTicketFillFromArray (
   PMTicketRef ticket,
   CFStringRef clientID,
   const PMTicketItemStruct *items,
   UInt32 itemCount
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

items
A pointer to an array of ticket item structures.

itemCount
The number of items in the `items` array.

Return Value
A result code. If the result code is anything other than `noErr`, it's possible that not all of the items were added successfully. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetAllocator

Obtains the allocator object used for allocating memory for a ticket.

```
OSStatus PMTicketGetAllocator (
   PMTicketRef ticket,
   CFAllocatorRef *allocator
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

allocator

On return, a reference to the `CFAllocator` object associated with the ticket.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetAPIVersion

Obtains the version of the application programming interface (API) used to created a ticket.

```
OSStatus PMTicketGetAPIVersion (
   PMTicketRef ticket,
   CFStringRef *apiVersion
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

apiVersion

On return, a reference to a `CFString` object that specifies the version of the API used to create the ticket.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetBoolean

Obtains data for a ticket item of type `Boolean`.

```
OSStatus PMTicketGetBoolean (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   Boolean *value
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain. See the Base Services documentation for a description of the `CFStringRef` data type.

value

On return, points to the `Boolean` value for the item specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetBytes

Obtains data for a ticket item that's an array of `UInt8` values.

```
OSStatus PMTicketGetBytes (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   UInt8 *data,
   UInt32 *size
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

data

On return, points to an array of data of type `UInt8` for the item specified by the `key` parameter.

size

On input, pass the size of the buffer pointed to by the `data` parameter. On return, points to the number of bytes in the array.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
If you don't know the size of the data buffer, you need to call the function `PMTicketGetBytes` twice. First, call the function to get the number of items in the array specified by the parameter `data`. You should call the function again after you allocate space for the array.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetCFArray

Obtains data for a ticket item that's a Core Foundation array.

```
OSStatus PMTicketGetCFArray (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFArrayRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item

On return, points to a Core Foundation array reference for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetCFBoolean

Obtains data for a ticket item that's an array of `CFBoolean` values.

```
OSStatus PMTicketGetCFBoolean (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFBooleanRef *item
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key
A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item
On return, points to a Core Foundation Boolean reference for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetCFData

Obtains data for a ticket item of type `CFData`.

```
OSStatus PMTicketGetCFData (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFDataRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item

On return, points to a Core Foundation data reference for the item specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetCFDate

Obtains data for a ticket item of type `CFDate`.

```
OSStatus PMTicketGetCFDate (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFDateRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item

On return, points to a Core Foundation date reference for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetCFDictionary

Obtains data for a ticket item of type `CFDictionary`.

```
OSStatus PMTicketGetCFDictionary (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFDictionaryRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item

On return, points to a Core Foundation dictionary reference for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetCFNumber

Obtains data for a ticket item of type `CFNumber`.

```
OSStatus PMTicketGetCFNumber (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFNumberRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item

On return, points to a Core Foundation number reference for the item specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetCFString

Obtains data for a ticket item of type `CFString`.

```
OSStatus PMTicketGetCFString (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFStringRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item

On return, points to a Core Foundation string reference for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetCString

Obtains data for a ticket item of that's a C-style string.

```
OSStatus PMTicketGetCString (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   UInt32 bufferSize,
   CFStringEncoding encoding,
   char *value
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

bufferSize

The size of the string specified by the `value` parameter.

encoding

The encoding of the string. You should supply one of the `CFStringBuiltInEncodings` constants defined in Core Foundation String Services.

value

On return, points to the C-style string for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetDouble

Obtains data for a ticket item of type `double`.

```
OSStatus PMTicketGetDouble (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   double *value
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key
A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

value
On return, points to the `double` value for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetEnumType

Obtains a ticket's ticket type (job, paper, converter, and so on).

```
OSStatus PMTicketGetEnumType (
   PMTicketRef ticket,
   PMTicketType *ticketType
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

ticketType

On return, points to a value of type `PMTicketType` that specifies a ticket's type.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetIndexPMResolution

Obtains an indexed resolution for a ticket item that's an array of type `PMResolution`.

```
OSStatus PMTicketGetIndexPMResolution (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   UInt32 index,
   PMResolution *res
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

index

The index of the resolution you want to obtain.

res

On return, points to the resolution (`PMResolution`) specified by the `index` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetItem

Obtains data for a ticket item of type `CFType`.

```
OSStatus PMTicketGetItem (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   CFTypeRef *item
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key
A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item
On return, point to a Core Foundation type reference for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetLockedState

Obtains the lock state of a ticket.

```
OSStatus PMTicketGetLockedState (
   PMTicketRef ticket,
   Boolean *lockedState
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

lockedState

On return, points to `true` if the ticket is locked and `false` if the ticket is unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
Tickets can be locked by the printing system. If you try to modify a locked ticket the result code `kPMTicketLocked` is returned. You can call the function `PMTicketGetLockedState` before you call any other function that modifies the ticket.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`PMTicketDeprecated.h`

PMTicketGetMetaItem

Obtains data for a item added with the function `PMTicketSetMetaItem`.

Not recommended.

```
OSStatus PMTicketGetMetaItem (
   PMTicketRef ticket,
   CFStringRef key,
   CFTypeRef *item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

item

On return, a reference to the data for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In
`PMTicketDeprecated.h`

PMTicketGetPMRect

Obtains data for a ticket item of type `PMRect`.

```
OSStatus PMTicketGetPMRect (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   PMRect *value
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key
A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

value
On return, points to the `PMRect` data for the item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetPMResolution

Obtains data for a ticket item of type `PMResolution`.

```
OSStatus PMTicketGetPMResolution (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   PMResolution *res
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

res

On return, points to the `PMResolution` data for the item specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetPPDDict

Obtains data for a ticket item that's a PostScript printer description (PPD) dictionary.

```
OSStatus PMTicketGetPPDDict (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFMutableDictionaryRef *dict
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

dict

On return, points to a reference to the PostScript printer description (PPD) dictionary associated with the specified ticket. The dictionary holds pairs of PPD main and option keywords. The main keywords are specified as keys in the dictionary and the options keywords specify the value.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`PMTicketDeprecated.h`

PMTicketGetPString

Obtains data for a ticket item that's a Pascal-style string.

```
OSStatus PMTicketGetPString (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   UInt32 bufferSize,
   CFStringEncoding encoding,
   StringPtr value
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

bufferSize

The size of the `value` parameter.

encoding

The encoding of the string. You should supply one of the `CFStringBuiltInEncodings` constants defined in Core Foundation String Services.

value

On return, points to the Pascal-style string value for the ticket item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetRetainCount

Obtains the retention count for a ticket.

```
OSStatus PMTicketGetRetainCount (
   PMTicketRef ticket,
   CFIndex *retainCount
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

retainCount
On return, points to the retention count for the ticket.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
The function `PMTicketGetRetainCount` behaves similarly to the Core Foundation function `CFGetRetainCount`.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetSInt32

Obtains data for a ticket item of type `SInt32`.

```
OSStatus PMTicketGetSInt32 (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   SInt32 *value
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1
Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

value

On return, points to the `SInt32` value for the ticket item specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetTicket

Obtains a subticket from a ticket.

```
OSStatus PMTicketGetTicket (
   PMTicketRef ticket,
   CFStringRef requestedType,
   UInt32 index,
   PMTicketRef *retrievedTicket
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

requestedType

A string that specifies the ticket type of the ticket you want to obtain. See "Ticket Type Strings" (page 1831) for a list of strings you can pass.

index

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

retrievedTicket

On return, points to the ticket reference whose type you specified.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketGetType

Obtains a value that specifies the ticket's type.

```
OSStatus PMTicketGetType (
   PMTicketRef ticket,
   CFStringRef *ticketType
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

ticketType

On return, points to a string value that specifies the ticket's type. See "PMTicketType" (page 1801) for a list of constants that can be returned

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketGetUInt32

Obtains an item of type `UInt32` from a ticket.

```
OSStatus PMTicketGetUInt32 (
   PMTicketRef ticket,
   UInt32 nodeIndex1,
   UInt32 nodeIndex2,
   CFStringRef key,
   UInt32 *value
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

nodeIndex1

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

nodeIndex2

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

key

A reference to a `CFString` object that uniquely identifies the ticket item whose value you want to obtain.

value

On return, points to `UInt32` value for the ticket item specified by the `key` parameter.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketIsItemLocked

Checks to see if an item in a ticket is locked.

```
OSStatus PMTicketIsItemLocked (
   PMTicketRef ticket,
   CFStringRef key,
   Boolean *locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to check.

locked
On return, points to `true` if the item is locked or `false` if the item is not locked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
The function `PMTicketIsItemLocked` checks only those items stored in the top-level of the ticket. It does not check items in subtickets.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`PMTicketDeprecated.h`

PMTicketLockItem

Locks an item in a ticket.

```
OSStatus PMTicketLockItem (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to lock.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Discussion

The function `PMTicketLockItem` locks only those items stored in the top-level of the ticket. It does not lock items in sub-tickets.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`PMTicketDeprecated.h`

PMTicketReadXMLFromFile

Restores a ticket previously converted to XML and saved as a file.

```
OSStatus PMTicketReadXMLFromFile (
   CFAllocatorRef allocator,
   const char *path,
   PMTicketRef *ticket,
   CFStringRef *errorString
);
```

Parameters

allocator

A reference to the allocator object to be used for allocating memory. Pass a reference to a valid allocator or `kCFAllocatorDefault` to request the default allocator.

path

A string that specifies the path name of the XML file you want to read.

ticket

A reference to a ticket created by calling the function `PMTicketCreate`. On return, the ticket contains the entries specified by the XML file.

errorString
On return, a reference to a `CFString` object that contains an error message if an error occurred.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketRelease

Decrements the retention count of a ticket object.

```
OSStatus PMTicketRelease (
   PMTicketRef ticket
);
```

Parameters
ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketReleaseAndClear

Decrements the retention count of a ticket and sets the ticket reference to `NULL`.

```
OSStatus PMTicketReleaseAndClear (
   PMTicketRef *ticket
);
```

Parameters
ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketReleaseItem

Removes an item from a ticket.

```
OSStatus PMTicketReleaseItem (
   PMTicketRef ticket,
   CFStringRef key
);
```

Parameters

ticket

A reference to a ticket created by calling the function PMTicketCreate.

key

A reference to a CFString object that uniquely identifies the ticket item you want to remove.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
An item can only be released if it is not locked. The function PMTicketReleaseItem works differently than the function PMTicketDeleteItem. The function PMTicketDeleteItem makes the item unavailable, but keeps information about the item in the ticket. The function PMTicketReleaseItem removes the item from the ticket.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketRemoveTicket

Removes a subticket from a ticket.

```
OSStatus PMTicketRemoveTicket (
   PMTicketRef ticket,
   CFStringRef typeToRemove,
   UInt32 index
);
```

Parameters

ticket

A reference to a ticket created by calling the function PMTicketCreate.

typeToRemove

A reference to a CFString object that specfies the ticket type of the ticket you want to create. See "Ticket Type Strings" (page 1831) for a list of constants you can pass.

index

Reserved for future use. Currently, you must pass the constant kPMTopLevel.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketRetain

Increments the retention count of a ticket object.

```
OSStatus PMTicketRetain (
   PMTicketRef ticket
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetBoolean

Writes an item of type `Boolean` to a ticket.

```
OSStatus PMTicketSetBoolean (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   Boolean value,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value
The `Boolean` value to which you want to set the ticket item entry specified by the `key` parameter.

locked
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetBytes

Writes an item that's an array of type `UInt8` to a ticket.

```
OSStatus PMTicketSetBytes (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   const UInt8 *data,
   UInt32 size,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

data
A pointer to the data to which you want to set the ticket item entry specified by the `key` parameter.

size
The size of the data, in bytes.

locked
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetCFArray

Writes an item of type `CFArray` to a ticket.

```
OSStatus PMTicketSetCFArray (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFArrayRef item,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item
A reference to the Core Foundation array to which you want to set the ticket item entry specified by the `key` parameter.

locked
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetCFBoolean

Writes an item of type `CFBoolean` to a ticket.

```
OSStatus PMTicketSetCFBoolean (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFBooleanRef item,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item

A reference to the `CFBoolean` value to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketSetCFData

Writes an item of type `CFData` to a ticket.

```
OSStatus PMTicketSetCFData (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFDataRef item,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item

A reference to the `CFData` to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetCFDate

Writes an item of type `CFDate` to a ticket.

```
OSStatus PMTicketSetCFDate (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFDateRef item,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item

A reference to the `CFDate` value to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketSetCFDictionary

Writes an item of type CFDictionary to a ticket.

```
OSStatus PMTicketSetCFDictionary (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFDictionaryRef item,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function PMTicketCreate.

clientID
A reference to a CFString object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as CFSTR("com.myvendorname.myprintingcode").

key
A reference to a CFString object that uniquely identifies the ticket item you want to set.

item
A reference to the Core Foundation dictionary to which you want to set the ticket item entry specified by the key parameter.

locked
Pass true to set the item to locked; false to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketSetCFNumber

Writes an item of type CFNumber to a ticket.

```
OSStatus PMTicketSetCFNumber (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFNumberRef item,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item

A reference to the `CFNumber` value to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketSetCFString

Writes an item of type `CFString` to a ticket.

```
OSStatus PMTicketSetCFString (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFStringRef item,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item

A reference to the `CFString` object to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetCString

Writes an item that's a C-style string to a ticket.

```
OSStatus PMTicketSetCString (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   const char *value,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value

The C-style string to which you want to set the ticket item entry specified by the `key` parameter. The string must use UTF-8 encoding.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketSetCStringArray

Writes an item that's an array of C-style strings to a ticket.

```
OSStatus PMTicketSetCStringArray (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   const char **cStringArray,
   UInt32 count,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function PMTicketCreate.

clientID
A reference to a CFString object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as CFSTR("com.myvendorname.myprintingcode").

key
A reference to a CFString object that uniquely identifies the ticket item you want to set.

cStringArray
The array of C-style strings to which you want to set the ticket item entry specified by the key parameter. The strings must use UTF-8 encoding.

count
The number of characters in the C-style string array.

locked
Pass true to set the item to locked; false to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
PMTicket.h

PMTicketSetDouble

Writes an item of type double to a ticket.

```
OSStatus PMTicketSetDouble (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   double value,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value

The `double` value to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketSetDoubleArray

Writes an item that's an array of values of type `double` to a ticket.

```
OSStatus PMTicketSetDoubleArray (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   const double *array,
   UInt32 count,
   Boolean changeable
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

doubleArray
A pointer to an array of `double` values to which you want to set the ticket item entry specified by the `key` parameter.

count
The number of values in the array specified by the `doubleArray` parameter.

changeable
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetItem

Writes an item of type `CFType` to a ticket.

```
OSStatus PMTicketSetItem (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   CFTypeRef item,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item
A reference to the generic Core Foundation data to which you want to set the ticket item entry specified by the `key` parameter.

locked
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetMetaItem

Writes an item that does not need to be stored in an XML-representation of a ticket.

Not recommended.

```
OSStatus PMTicketSetMetaItem (
   PMTicketRef ticket,
   CFStringRef key,
   CFTypeRef item
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item

A reference to the generic Core Foundation data to which you want to set the ticket item entry specified by the `key` parameter.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
You can use the function `PMTicketSetMetaItem` to add an item to a ticket when you don't want that item o be written to an XML-representation of the ticket. In other words, when you want to temporarily add an item to a ticket. Items added with this function cannot be locked.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`PMTicketDeprecated.h`

PMTicketSetPMRect

Writes an item of type `PMRect` to a ticket.

```
OSStatus PMTicketSetPMRect (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   PMRect *value,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value

A pointer to the `PMRect` value to which you want to set the ticket item entry specified by the `key` parameter. A `PMRect` data type is an array of four `double` values.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketSetPMRectArray

Writes an item that's an array of values of type `PMRect` to a ticket.

```
OSStatus PMTicketSetPMRectArray (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   PMRect *pmRectArray,
   UInt32 count,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

pmRectArray

A pointer to an array of `PMRect` values to which you want to set the ticket item entry specified by the `key` parameter. A `PMRect` data type is an array of four `double` values.

count

The number of values in the array specified by the `pmRectArray` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetPMResolution

Writes an item of type `PMResolution` to a ticket.

```
OSStatus PMTicketSetPMResolution (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   PMResolution *value,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value

A pointer to the `PMResolution` value to which you want to set the ticket item entry specified by the `key` parameter. A `PMResolution` data type is an array of two `double` values.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetPMResolutionArray

Writes an item that's an array of data of type `PMResolution` to a ticket.

```
OSStatus PMTicketSetPMResolutionArray (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   PMResolution *pmResolutionArray,
   UInt32 count,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

pmResolutionArray
A pointer to an array of the `PMResolution` values to which you want to set the ticket item entry specified by the `key` parameter. A `PMResolution` data type is an array of two `double` values.

count
The number of items in the array specified by the `pmResolutionArray` parameter.

locked
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetPString

Writes an item that's a Pascal-style string to a ticket.

```
OSStatus PMTicketSetPString (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   ConstStringPtr value,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value

A pointer to the Pascal-style string to which you want to set the ticket item entry specified by the `key` parameter. The string must use MacRoman encoding.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketSetSInt32

Writes an item of type `SInt32` to a ticket.

```
OSStatus PMTicketSetSInt32 (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   SInt32 value,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value

The `SInt32` value to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetSInt32Array

Writes an item that's an array of data of type `SInt32` to a ticket.

```
OSStatus PMTicketSetSInt32Array (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   const SInt32 *sInt32Array,
   UInt32 count,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

sInt32Array

A pointer to an array of `SInt32` values to which you want to set the ticket item entry specified by the `key` parameter.

count

The number of values in the array specified by the `sInt32Array` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetTemplate

Writes an item that's a job template to a ticket.

```
OSStatus PMTicketSetTemplate (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   PMTemplateRef item,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

item
A reference to a job template created by calling the function `PMTemplateCreate`.

locked
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetTicket

Writes a subticket to a ticket.

```
OSStatus PMTicketSetTicket (
   PMTicketRef ticket,
   PMTicketRef ticketToAdd,
   UInt32 index
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`. This is the ticket to which you want to add a subticket. Any ticket can contain another ticket.

ticketToAdd

A reference to a ticket created by calling the function `PMTicketCreate`. This is the ticket you want to be a subticket.

index

Reserved for future use. Currently, you must pass the constant `kPMTopLevel`.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketSetUInt32

Writes an item of type `UInt32` to a ticket.

```
OSStatus PMTicketSetUInt32 (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   UInt32 value,
   Boolean locked
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

value

The `UInt32` value to which you want to set the ticket item entry specified by the `key` parameter.

locked

Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketSetUInt32Array

Writes an item that's an array of data of type `UInt32` to a ticket.

```
OSStatus PMTicketSetUInt32Array (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key,
   const UInt32 *uInt32Array,
   UInt32 count,
   Boolean locked
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

clientID
A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key
A reference to a `CFString` object that uniquely identifies the ticket item you want to set.

uInt32Array
A pointer to the array of `UInt32` values to which you want to set the ticket item entry specified by the `key` parameter.

count
The number of values in the array specified by the `uInt32Array` parameter.

locked
Pass `true` to set the item to locked; `false` to set it to unlocked.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketToXML

Converts the data in a ticket to XML.

```
OSStatus PMTicketToXML (
   PMTicketRef ticket,
   CFDataRef *anXMLTicket
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

anXMLTicket

On return, points to a Core Foundation data reference that represents job template data. You are responsible for releasing the `CFData` reference.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Discussion

If you want to write the XML data is to a file, use the function `PMTicketWriteXMLToFile`.

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketUnlockItem

Unlocks an item in a ticket.

```
OSStatus PMTicketUnlockItem (
   PMTicketRef ticket,
   CFStringRef clientID,
   CFStringRef key
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

clientID

A reference to a `CFString` object that uniquely identifies your application. The string should be in a format similar to a Java-style package name (think of it as a reverse URL), such as `CFSTR("com.myvendorname.myprintingcode")`.

key

A reference to a `CFString` object that uniquely identifies the ticket item you want to unlock.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
The function `PMTicketUnlockItem` unlocks only those items stored in the top-level of the ticket. It does not unlock items in subtickets.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`PMTicketDeprecated.h`

PMTicketValidate

Validates a ticket against the constraint values specified in a job template.

```
OSStatus PMTicketValidate (
   PMTicketRef ticket,
   PMTemplateRef verifyingTemplate,
   CFArrayRef *invalidItems
);
```

Parameters

ticket
A reference to a ticket created by calling the function `PMTicketCreate`.

verifyingTemplate
A reference to the job template against which you want to validate the ticket.

invalidItems
On return, points to an array of invalid items, should there be any. If there are no invalid items, and the function returns `noErr`, then the value of `invalidItems` is undefined.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Discussion
Only those items in the ticket that have corresponding entries in the job template are checked. In other words, a ticket item's key must match a template item's key for the ticket item to be validated.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

PMTicketWriteXML

Converts a ticket to XML and then writes it to a file stream.

```
OSStatus PMTicketWriteXML (
   PMTicketRef ticket,
   FILE *file
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

xmlFile

On input, the file to which you want the XML data to be written.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMTicketWriteXMLToFile

Converts a ticket to XML and then writes it to a file.

```
OSStatus PMTicketWriteXMLToFile (
   PMTicketRef ticket,
   const char *path
);
```

Parameters

ticket

A reference to a ticket created by calling the function `PMTicketCreate`.

path

On input, the path of the file to which you want the XML data to be written. The function opens the file or creates one if one doesn't exist.

Return Value

A result code. See "Ticket Services Result Codes" (page 1832).

Availability

Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In

`PMTicket.h`

PMXMLToTicket

Converts a ticket saved as XML to a ticket.

```
OSStatus PMXMLToTicket (
   CFAllocatorRef allocator,
   CFDataRef anXMLTicket,
   PMTicketRef *ticket,
   CFStringRef *conversionError
);
```

Parameters

allocator

A reference to the allocator object to be used for allocating memory. Pass a reference to a valid allocator or `kCFAllocatorDefault` to request the default allocator.

anXMLTicket

A reference to Core Foundation data that contains previously-converted job template data.

ticket

On return, a reference to a ticket that contains the data converted from the XML data specified by the `anXMLTicket` parameter.

conversionError

On return, a reference to a `CFString` object that specifies errors during the conversion process, if any. Pass `NULL` if you are not interested in getting the errors.

Return Value
A result code. See "Ticket Services Result Codes" (page 1832).

Availability
Not available in CarbonLib.

Available in Mac OS X 10.0 and later.

Declared In
`PMTicket.h`

Data Types

ConstCStrList

Represents a static list of C-string pointers.

```
typedef CStrList ConstCStrList;
```

Discussion
For more information see "CStrList" (page 1798).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

ConstPMRectList

Represents a static list of `PMRect` data structures.

```
typedef PMRectList ConstPMRectList;
```

Discussion
For more information see "PMRectList" (page 1799).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

ConstSInt32List

Represents a static list of `SInt32List` data structures.

```
typedef SInt32List ConstSInt32List;
```

Discussion
Form more information see "SInt32List" (page 1801).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

CStrList

Contains an array of `CString` pointers and the number of pointers in the array.

```
struct CStrList {
    SInt32 count;
    char **strArray;
};
typedef struct CStrList CStrList;
```

Fields
`count`
The number of `CString` pointers in the array.
`strArray`
A pointer to an array of `CString` pointers.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

PMPrintingPhaseType

Represents printing phases.

```
typedef UInt16 PMPrintingPhaseType;
```

Discussion
For more information see "Printing Phase Types" (page 1824).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`PMTicket.h`

PMRectList

Contains a list of `PMRect` data structures.

```
struct PMRectList {
    SInt32 count;
    PMRect **pmRectArray;
};
typedef struct PMRectList PMRectList;
```

Fields

`count`
The number of `PMRect` pointers in the array.

`pmRectArray`
A pointer to a list of `PMRect` data structures.

Discussion
A `PMRect` data structure contains a set of four `double` values (top, left, bottom, and right). This structure is used to specify page and paper rectangles.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

PMTemplateRef

Refers to a template object that contains private variables and functions necessary to represent a job template.

```
typedef struct OpaquePMTemplateRef* PMTemplateRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

PMTicketErrors

Represents values that indicate error conditions.

```
typedef SInt16 PMTicketErrors;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

PMTicketItemStruct

Contains information about a ticket item.

```
struct PMTicketItemStruct {
    char *key;
    PMTicketItemType itemType;
    Boolean locked;
    union {
        const   void *GenericData;
    const   char *cString;
      SInt32 sInt32;
      UInt32 boolean;
      ConstCStrList *cStrlist;
      PMRect *rect;
      ConstSInt32List *sInt32List;
      ConstPMRectList *pmRectList;
    } value;
};
```

Fields

`key`
A string that uniquely identifies the item.

`itemType`
The type of item defined in the union.

`locked`
The lock state of the item.

`value`
The data associated with the item.

Discussion
You can use this structure to define a static ticket item. An array of these structures can then be converted to a ticket by calling the function `PMTicketFillFromArray`.

PMTicketItemType

Represents a ticket item type.

```
typedef UInt16 PMTicketItemType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

PMTicketRef

Refers to a ticket object that contains private variables and functions necessary to represent a ticket.

```
typedef struct OpaquePMTicketRef* PMTicketRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

PMTicketType

Represents a ticket type.

```
typedef SInt16 PMTicketType;
```

Discussion
For more information see "PMTicketType" (page 1801).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTicket.h`

PMValueType

Represents a value type.

```
typedef SInt32 PMValueType;
```

Discussion
For more information see "Item Value Types" (page 1808).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMTemplate.h`

SInt32List

Contains an array of `SInt32` values.

```
struct SInt32List {
    SInt32 count;
    const SInt32 *sInt32Array;
};
typedef struct SInt32List SInt32List;
```

Fields

`count`

The number of signed 32-bit values in the array.

`sInt32Array`

A pointer to the array.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PMTicket.h`

Constants

ColorSync Options

Defines strings and keys for items related to ColorSync options.

```
#define kPMColorDeviceIDStr kPMPrinterInfoPrelude "PMColorDeviceID"
#define kPMColorDeviceIDKey CFSTR(kPMColorDeviceIDStr)
#define kPMColorSyncProfilesStr kPMPrinterInfoPrelude "PMColorSyncProfiles"
#define kPMColorSyncProfilesKey CFSTR(kPMColorSyncProfilesStr)
```

Constants

`kPMColorDeviceIDStr`

The color device ID string.

`kPMColorDeviceIDKey`

The value of this key is a `CFString` representing a `CFUUID`; it must be unique per device.

`kPMColorSyncProfilesStr`

The ColorSync profiles sting.

`kPMColorSyncProfilesKey`

The value of this key is a `CFArray` of `CFDictionary` data structures, one `CFDictionary` data structure for each factory profile.

Constraint Types

Specify a constraint type for a job template entry.

```
typedef SInt32 PMConstraintType;
enum {
    kPMConstraintUndefined = 0,
    kPMConstraintRange = 1,
    kPMConstraintList = 2,
    kPMConstraintPrivate = 3
};
```

Constants

`kPMConstraintUndefined`
Values are not constrained.
Available in Mac OS X v10.0 and later.
Declared in `PMTemplate.h`.

`kPMConstraintRange`
Values are constrained by a range of two values, both of type `CFTypeRef`.
Available in Mac OS X v10.0 and later.
Declared in `PMTemplate.h`.

`kPMConstraintList`
Values are constrained by an array.
Available in Mac OS X v10.0 and later.
Declared in `PMTemplate.h`.

`kPMConstraintPrivate`
Values are constrained privately, and should not be checked by the system.
Available in Mac OS X v10.0 and later.
Declared in `PMTemplate.h`.

Converter Setup Ticket Keys

Defines strings and keys for items related to a converter setup ticket.

```
#define kPMConverterSetupPrelude "com.apple.print.ConverterSetup."
#define kPMBandingRequestedStr kPMConverterSetupPrelude "PMBandingRequested"
#define kPMBandingRequestedKey CFSTR(kPMBandingRequestedStr)
#define kPMRequiredBandHeightStr kPMConverterSetupPrelude "PMRequiredBandHeight"
#define kPMRequiredBandHeightKey CFSTR(kPMRequiredBandHeightStr)
#define kPMDepthSwitchingEnabledStr kPMConverterSetupPrelude "PMDepthSwitching"
#define kPMDepthSwitchingEnabledKey CFSTR(kPMDepthSwitchingEnabledStr)
#define kPMWhiteSkippingEnabledStr kPMConverterSetupPrelude "PMWhiteSpaceSkipping"
#define kPMWhiteSkippingEnabledKey CFSTR(kPMWhiteSkippingEnabledStr)
#define kPMConverterResHorizontalStr kPMConverterSetupPrelude
"PMConversionResHorizontal"
#define kPMConverterResHorizontalKey CFSTR(kPMConverterResHorizontalStr)
#define kPMConverterResVerticalStr kPMConverterSetupPrelude "PMConversionResVertcial"
#define kPMConverterResVerticalKey CFSTR(kPMConverterResVerticalStr)
#define kPMRequestedPixelFormatStr kPMConverterSetupPrelude "PMPixelFormat"
#define kPMRequestedPixelFormatKey CFSTR(kPMRequestedPixelFormatStr)
#define kPMRequestedPixelLayoutStr kPMConverterSetupPrelude "PMPixelLayout"
#define kPMRequestedPixelLayoutKey CFSTR(kPMRequestedPixelLayoutStr)
#define kPMCVColorSyncProfileIDKey CFSTR(kPMCVColorSyncProfileIDStr)
```

Constants

`kPMConverterSetupPrelude`
The converter-setup prelude sting.

`kPMBandingRequestedStr`
The banding-requested string.

`kPMBandingRequestedKey`
The value of this key is a `CFBoolean`; turns banding on if it's available.

`kPMRequiredBandHeightStr`
The banding-height string.

`kPMRequiredBandHeightKey`
The value of this key is a `CFNumber` value; specifies the number of scan lines needed for each band. If it specifies the whole page, banding is disabled.

`kPMDepthSwitchingEnabledStr`
The depth-switching-enabled string.

`kPMDepthSwitchingEnabledKey`
The value of this key is a `CFBoolean` value; `true` specifies the printer module requests the converter to switch between black & white and color bands when possible.

`kPMWhiteSkippingEnabledStr`
The white-skipping-enabled string.

`kPMWhiteSkippingEnabledKey`
The value of this key is a `CFBoolean`; `true` specifies the printer module requests the converter to skip over white space if possible.

`kPMConverterResHorizontalStr`
The horizontal rendering resolution string.

`kPMConverterResHorizontalKey`
The value of this key is a `CFNumber` of type `CFNumberDoubleType`; specifies the final horizontal rendering resolution.

`kPMConverterResVerticalStr`
The vertical resolution string.

`kPMConverterResVerticalKey`
The value of this key is a `CFNumber` of type `CFNumberDoubleType`; specifies the final vertical rendering resolution.

`kPMRequestedPixelFormatStr`
The pixel format string.

`kPMRequestedPixelFormatKey`
The value of this key is a `CFNumber` of type `CFNumberLongType`; specifies the pixel format requested of the converter.

`kPMRequestedPixelLayoutStr`
The pixel layout string.

`kPMRequestedPixelLayoutKey`
The value of this key is a `CFNumber` of type `CFNumberLongType`; specifies the pixel layout requested of the converter.

`kPMCVColorSyncProfileIDKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies the profile ID for the ColorSync profile to be used with the print job.

Data Transmission Keys

Defines strings and keys for items related to data transmission.

```
#define kPMIsBinaryOKStr            kPMPrinterInfoPrelude "PMIsBinaryOK"
#define kPMIsBinaryOKKey            CFSTR(kPMIsBinaryOKStr)
#define kPM8BitCommStr            kPMPrinterInfoPrelude "PM8BitComm"
#define kPM8BitCommKey            CFSTR(kPM8BitCommStr)
#define kPMTransparentCommStr           kPMPrinterInfoPrelude "PMTransparentComm"
#define kPMTransparentCommKey            CFSTR(kPMTransparentCommStr)
```

Constants

`kPMIsBinaryOKStr`
The binary is okay string.

`kPMIsBinaryOKKey`
The value of this key is a `CFBoolean` representing the result of querying the PostScript printer about its ability to accept binary data. It is possible for the underlying communications to support binary data, both high bit characters and control characters, but for a spooler on the other end of the channel to not accept binary data. This value represents the spooler/printer's abilities.

`kPM8BitCommStr`
The 8-bit communication string.

`kPM8BitCommKey`
The value of this key is a `CFBoolean` indicating whether the communications channel can transmit characters in the range 0x80 -)xFF inclusive without them being damaged or interpreted as channel control characters.

`kPMTransparentCommStr`
The transparent communication string.

`kPMTransparentCommKey`
The value of this key is a `CFBoolean` indicating whether the communications channel can transmit characters in the range 0x00 - 0x1F inclusive without them being damaged or interpreted as channel control characters.

Document Ticket Keys

Defines strings and keys for items in a document ticket; currently not used.

```
#define kPMDocumentTicketPrelude "com.apple.print.DocumentTicket."
#define kPMSpoolFormatStr kPMDocumentTicketPrelude "PMSpoolFormat"
#define kPMSpoolFormatKey CFSTR(kPMSpoolFormatStr)
#define kPMPrinterModuleFormatStr kPMDocumentTicketPrelude "PMDocPMInputFormat"
#define kPMPrinterModuleFormatKey CFSTR(kPMPrinterModuleFormatStr)
```

Constants

`kPMDocumentTicketPrelude`
Currently not used.

`kPMSpoolFormatStr`
Currently not used.

`kPMSpoolFormatKey`
Currently not used.

`kPMPrinterModuleFormatStr`
Currently not used.

`kPMPrinterModuleFormatKey`
Currently not used.

Drawing Resolution Keys

Defines strings and keys for items in related to drawing resolution.

```
#define kPMPrinterSuggestedResStr kPMPrinterInfoPrelude
"PMPrinterSuggestedRes"
#define kPMPrinterSuggestedResKey CFSTR(kPMPrinterSuggestedResStr
        #define kPMPrinterMinResStr kPMPrinterInfoPrelude "PMPrinterMinRes"
#define kPMPrinterMinResKey CFSTR(kPMPrinterMinResStr)
#define kPMPrinterMaxResStr kPMPrinterInfoPrelude "PMPrinterMaxRes"
#define kPMPrinterMaxResKey CFSTR(kPMPrinterMaxResStr)
```

Constants

`kPMPrinterSuggestedResStr`
The suggested application drawing resolutions string.

`kPMPrinterSuggestedResKey`
The value of this key is the suggested application drawing resolutions key.

`kPMPrinterMinResStr`
The minimum drawing resolution string.

`kPMPrinterMinResKey`
The value of this key is the minimum range of resolutions for the printer; specified as a `CFArray` of two `CFNumber` values of type `kCFNumberDoubleType`.

`kPMPrinterMaxResStr`
The maximum printer resolution string.

`kPMPrinterMaxResKey`
The value of this key is the maximum range of resolutions for the printer; specified as a `CFArray` of two `CFNumber` values of type `kCFNumberDoubleType`.

Duplex Options

Specify values for the duplex options key (`kPMDuplexingKey`.)

```
enum {
    kPMDuplexNone = 1,
    kPMDuplexNoTumble = 2,
    kPMDuplexTumble = 3,
    kPMSimplexTumble = 4,
    kPMDuplexDefault = 1
};
```

Constants

`kPMDuplexNone`
Don't use duplex.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMDuplexNoTumble`
Print on both sides of the paper; flip pages from left to right.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMDuplexTumble`
Print on both sides of the paper; tumbling so pages flip top to bottom.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMSimplexTumble`
Print on only one side of the paper, but tumble the images while printing.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMDuplexDefault`
Don't use duplex; this option is the same as `kPMDuplexNone`.

Error Handling Options

Specify whether or not an error handler is available.

```
enum {
    kPSNoErrorHandler = 0,
    kPSErrorHandler = 1
};
```

Constants

`kPSNoErrorHandler`
Specifies that an error handler is not available.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPSErrorHandler`
Specifies that an error handler is available.

Available in Mac OS X v10.0 and later.

Declared in `PMTicket.h`.

Fetch options

Specifies whether an item should be fetched.

```
#define kPMDontFetchItem NULL
```

Constants

`kPMDontFetchItem`
Don't fetch the item.

Installable Options

Defines a string and key for installable printer options.

```
#define kPMInstallableOptionStr kPMPrinterInfoPrelude
"PMInstallableOption"
#define kPMInstallableOptionKey CFSTR(kPMInstallableOptionStr)
```

Constants

`kPMInstallableOptionStr`
The installable options string.

`kPMInstallableOptionKey`
The value of this key is a packed array of Pascal strings that specifies the installable options in the PostScript printer description (PPD) file. The strings are as key-value pairs of PPD main and option keywords.

Item Value Types

Specify the data type of a ticket or template item.

```
typedef UInt16 PMTicketItemType;
enum {
    kPMItemInvalidType = 0,
    kPMItemCStringType = 1,
    kPMItemSInt32Type = 2,
    kPMItemBooleanType = 3,
    kPMItemCStrListType = 4,
    kPMItemPMRectType = 5,
    kPMItemSInt32ListType = 6,
    kPMItemPMRectListType = 7
};
```

Constants

`kPMItemInvalidType`
The type is not valid.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMItemCStringType`
A C-style string pointer.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMItemSInt32Type`
A signed 32-bit integer.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMItemBooleanType`
A `Boolean` value.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMItemCStrListType`
A list of C-style strings
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMItemPMRectType`
A pointer to a `PMRect` data structure.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMItemSInt32ListType`
A pointer to an `SInt32List` data structure.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMItemPMRectListType`
A pointer to a `PMRectList` data structure.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

Job Ticket Keys

Defines strings and keys for items in a job ticket.

```
#define kPMJobTicketPrelude "com.apple.print.JobInfo."
#define kPMJobNameStr kPMJobTicketPrelude "PMJobName"
#define kPMJobNameKey CFSTR(kPMJobNameStr)
#define kPMApplicationNameStr kPMJobTicketPrelude "PMApplicationName"
#define kPMApplicationNameKey CFSTR(kPMApplicationNameStr)
#define kPMUserLanguageStr kPMJobTicketPrelude "PMUserLanguage"
#define kPMUserLanguageKey CFSTR(kPMUserLanguageStr)
#define kPMJobOwnerStr kPMJobTicketPrelude "PMJobOwner"
#define kPMJobOwnerKey CFSTR(kPMJobOwnerStr)
#define kPMJobTemplateStr kPMJobTicketPrelude "PMJobTemplate"
#define  CFSTR(kPMJobTemplateStr)
#define kPMPhaseStr kPMJobTicketPrelude "PMPrintingPhase"
#define kPMPhaseKey CFSTR(kPMPhaseStr)
#define kPMOutputTypeStr kPMJobTicketPrelude "PMOutputType"
#define kPMOutputTypeKey CFSTR(kPMOutputTypeStr)
```

Constants

`kPMJobTicketPrelude`
The job ticket prelude string.

`kPMJobNameStr`
The job name string.

`kPMJobNameKey`
The value of this key is a `CFString` that specifies the name of the job to be displayed in the print queue dialog.

`kPMApplicationNameStr`
The application name string.

`kPMApplicationNameKey`
The value of this key is a `CFString` that specifies the application's name.

`kPMUserLanguageStr`
The user language string.

`kPMUserLanguageKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`.

`kPMJobOwnerStr`
The job owner string.

`kPMJobOwnerKey`
The value of this key is a `CFString` that specifies the name of the user who submitted the job.

`kPMJobTemplateStr`
The job template string.

`kPMJobTemplateKey`
The value of this key is a `CFDictionary` that is actually a `PMTemplateRef`.

`kPMPhaseStr`
The printing phase string.

`kPMPhaseKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration—Spooling, RIPing, and so forth.

`kPMOutputTypeStr`
The output type string.

`kPMOutputTypeKey`
The value of this key is a `CFString` that specifies a Mime type from the `kPMOutputTypeListKey` array generated by the printer module.

Default Copy/Collate Value

Specifies the default value for copy/collate.

```
enum {
    kPMCopyCollateDefault = 1
};
```

Constants

`kPMCopyCollateDefault`
Use the default value for copy/collate.

List Ticket Keys

Defines strings and keys for items in a list ticket.

```
#define kPMTicketListPrelude "com.apple.print.TicketList."
```

Constants

`kPMTicketListPrelude`
The list ticket prelude string.

Lock State

Specifies whether an item is locked or unlocked.

```
enum {
    kPMUnlocked = 0,
    kPMLocked = 1
};
```

Constants

`kPMUnlocked`
Indicates items is unlocked.

Available in Mac OS X v10.2 and later.

Declared in `PMDefinitions.h`.

`kPMLocked`
Indicates items is locked.

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

Memory Keys

Defines strings and keys for items related to printer memory.

```
#define kPMTotalMemInstalledStr kPMPrinterInfoPrelude
"PMTotalMemInstalled" #define kPMTotalMemInstalledKey
CFSTR(kPMTotalMemInstalledStr)
#define kPMTotalMemAvailableStr kPMPrinterInfoPrelude "PMTotalMemAvailable"
#define kPMTotalMemAvailableKey CFSTR(kPMTotalMemAvailableStr)
```

Constants

`kPMTotalMemInstalledStr`
The memory installed string.

`kPMTotalMemInstalledKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies the memory installed in the printer.

`kPMTotalMemAvailableStr`
The total memory available string.

`kPMTotalMemAvailableKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies the remaining memory available for use.

Page Format Ticket Keys

Defines strings and keys for items in a page format ticket.

```
#define kPMPageFormatPrelude "com.apple.print.PageFormat."
#define kPMAdjustedPaperRectStr kPMPageFormatPrelude "PMAdjustedPaperRect"
#define kPMAdjustedPaperRectKey CFSTR(kPMAdjustedPaperRectStr)
#define kPMAdjustedPageRectStr kPMPageFormatPrelude "PMAdjustedPageRect"
#define kPMAdjustedPageRectKey CFSTR(kPMAdjustedPageRectStr)
#define kPMDrawingResHorizontalStr kPMPageFormatPrelude "PMHorizontalRes"
#define kPMDrawingResHorizontalKey CFSTR(kPMDrawingResHorizontalStr)
#define kPMDrawingResVerticalStr kPMPageFormatPrelude "PMVerticalRes"
#define kPMDrawingResVerticalKey CFSTR(kPMDrawingResVerticalStr)
#define kPMPageScalingHorizontalStr kPMPageFormatPrelude "PMScaling"
#define kPMPageScalingHorizontalKey CFSTR(kPMPageScalingHorizontalStr)
#define kPMPageScalingVerticalStr kPMPageFormatPrelude "PMVerticalScaling"
#define kPMPageScalingVerticalKey CFSTR(kPMPageScalingVerticalStr)
#define kPMPageOrientationStr kPMPageFormatPrelude "PMOrientation"
#define kPMPageOrientationKey CFSTR(kPMPageOrientationStr)
#define kPMPageBackupRecordHdlStr kPMPageFormatPrelude "BackupPrintRecordHandle"
#define kPMPageBackupRecordHdlKey CFSTR(kPMPageBackupRecordHdlStr)
#define kPMPageBackupRecordDataStr kPMPageFormatPrelude "BackupPrintRecord"
#define kPMPageBackupRecordDataKey CFSTR kPMPageBackupRecordDataStr)
#define kPMPageCustomDialogHdlStr kPMPageFormatPrelude "CustomDialogRecord"
#define kPMPageCustomDialogHdlKey CFSTR(kPMPageCustomDialogHdlStr)
#define kPMFormattingPrinterStr kPMPageFormatPrelude "FormattingPrinter"
#define kPMFormattingPrinterKey CFSTR(kPMFormattingPrinterStr)
```

Constants

`kPMPageFormatPrelude`
The page format ticket prelude string.

`kPMAdjustedPaperRectStr`
The adjusted paper rectangle string.

`kPMAdjustedPaperRectKey`
The value of this key is a `CFArray` of four `CFNumbers` of type `kCFNumberDoubleType` for the adjusted paper rectangle, in points.

`kPMAdjustedPageRectStr`
The adjusted page rectangle string.

`kPMAdjustedPageRectKey`
The value of this key is a `CFArray` of four `CFNumbers` of type `kCFNumberDoubleType` for the adjusted page rectangle, in points.

`kPMDrawingResHorizontalStr`
The horizontal drawing resolution string.

`kPMDrawingResHorizontalKey`
The value of this key is a `CFNumber` of type `kCFNumberDoubleType`; the drawing resolution in horizontal direction.

`kPMDrawingResVerticalStr`
The vertical drawing resolution string.

`kPMDrawingResVerticalKey`
The value of this key is a `CFNumber` of type `kCFNumberDoubleType`; the drawing resolution in vertical direction.

`kPMPageScalingHorizontalStr`
The horizontal page scaling string.

`PMPageScalingHorizontalKey`
The value of this key is a `CFNumber` of type `kCFNumberDoubleType`; the horizontal scaling factor applied to original page size.

`kPMPageScalingVerticalStr`
The vertical page scaling string.

`kPMPageScalingVerticalKey`
The value of this key is a `CFNumber` of type `kCFNumberDoubleType`; the vertical scaling factor applied to original page size.

`kPMPageOrientationStr`
The page orientation string.

`kPMPageOrientationKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; `PMOrientation`, 1 = portrait, 2 = landscape, 3 = reverse portrait, 4 = reverse landscape.

`kPMPageBackupRecordHdlStr`
The page backup record handle string.

`kPMPageBackupRecordHdlKey`
The value of this key is a print record handle (`CFData`); not used when data is flattened. (Mac OS 8 and 9 only)

`kPMPageBackupRecordDataStr`
The page backup record data string.

`kPMPageBackupRecordDataKey`
The value of this key is a print record (`CFData`) stored in complete form; used when flattening ticket with the print record. ((Mac OS 8 and 9 only)

`kPMPageCustomDialogHdlStr`
The page custom dialog handle string.

`kPMPageCustomDialogHdlKey`
: The value of this key is a handle (`CFData`) to the print record using for custom dialog calls; not stored when flattened. (Mac OS 8 and 9 only)

`kPMFormattingPrinterStr`
: The formatting printer string.

`kPMFormattingPrinterKey`
: The value of this key is a `CFString` that specifies name of the formatting printer.

Page Ticket Key

Defines strings and keys for items in a page ticket; currently not used.

```
#define kPMPageTicketPrelude "com.apple.print.PageTicket."
```

Constants

`kPMPageTicketPrelude`
: Currently not used.

Paper Info Ticket Keys

Defines strings and keys for items in a paper info ticket.

```
#define kPMPaperInfoPrelude "com.apple.print.PaperInfo."
#define kPMPaperNameStr kPMPaperInfoPrelude "PMPaperName"
#define kPMPaperNameKey CFSTR(kPMPaperNameStr)
#define kPMUnadjustedPaperRectStr kPMPaperInfoPrelude "PMUnadjustedPaperRect"
#define kPMUnadjustedPaperRectKey CFSTR(kPMUnadjustedPaperRectStr)
#define kPMUnadjustedPageRectStr kPMPaperInfoPrelude "PMUnadjustedPageRect"
#define kPMUnadjustedPageRectKey CFSTR(kPMUnadjustedPageRectStr)
#define kPMMatchPaperStr kPMPaperInfoPrelude "PMMatchPaper"
#define kPMMatchPaperKey CFSTR(kPMMatchPaperStr)
```

Constants

`kPMPaperInfoPrelude`
: The paper info ticket prelude string.

`kPMPaperNameStr`
: The paper name string.

`kPMPaperNameKey`
: The paper name key specifies the name of the paper displayed in the user interface.

`kPMUnadjustedPaperRectStr`
: The unadjusted paper rectangle string.

`kPMUnadjustedPaperRectKey`
: The unadjusted paper rectangle key specifies the unadjusted paper rectangle.

`kPMUnadjustedPageRectStr`
: The unadjusted page rectangle string.

`kPMUnadjustedPageRectKey`
: The unadjusted page rectangle key specifies the unadjusted page rectangle.

`kPMMatchPaperStr`
: The paper match string.

`kPMMatchPaperKey`
The paper match key specifies how closely the printing system must match the specified paper.

PostScript Language Level Targets

Specify language-level targets for PostScript printing.

```
enum {
    kPMPSTargetLanguageLevel2and3 = -3,
    kPMPSTargetLanguageLevel1and2 = -2,
    kPMPSTargetLanguageLevelUnknown = -1,
    kPMPSTargetLanguageLevel1 = 1,
    kPMPSTargetLanguageLevel2 = 2,
    kPMPSTargetLanguageLevel3 = 3,
    kPMPSTargetLanguageLevelDefault = -1
};
```

Constants

`kPMPSTargetLanguageLevel2and3`
Level 2 compatible, may take advantage of Level 3 features.

Available in Mac OS X v10.1 and later.

Declared in `PMTicket.h`.

`kPMPSTargetLanguageLevel1and2`
Level 1 compatible, may take advantage of Lever 2 and 3 features.

Available in Mac OS X v10.1 and later.

Declared in `PMTicket.h`.

`kPMPSTargetLanguageLevelUnknown`
Language level of target is unknown.

Available in Mac OS X v10.1 and later.

Declared in `PMTicket.h`.

`kPMPSTargetLanguageLevel1`
Level 1.

Available in Mac OS X v10.1 and later.

Declared in `PMTicket.h`.

`kPMPSTargetLanguageLevel2`
Level 2.

Available in Mac OS X v10.1 and later.

Declared in `PMTicket.h`.

`kPMPSTargetLanguageLevel3`
Level 3.

Available in Mac OS X v10.1 and later.

Declared in `PMTicket.h`.

`kPMPSTargetLanguageLevelDefault`
Same as `kPMPSTargetLanguageLevelUnknown`.

Available in Mac OS X v10.1 and later.

Declared in `PMTicket.h`.

PostScript Printer Description Tags

Defines strings and keys for items related to PostScript printer description files.

```
#define kPMDescriptionFileStr kPMPrinterInfoPrelude
"PMDescriptionFile"
#define kPMDescriptionFileKey CFSTR(kPMDescriptionFileStr)
#define kPMCompiledPPDStr kPMPrinterInfoPrelude "PMCompiledPPD"
#define kPMCompiledPPDKey CFSTR(kPMCompiledPPDStr)
```

Constants

`kPMDescriptionFileStr`
The printer description file string.

`kPMDescriptionFileKey`
The value of this key is a `CFString`; specifies the PostScript printer description file name or other description file.

`kPMCompiledPPDStr`
The compiled PostScript printer description string.

`kPMCompiledPPDKey`
The value of this key is a compiled PostScript printer description (`CFData`).

PostScript Printer Driver Keys

Defines a string and key for a PostScript printer driver.

```
#define kPMPrinterIsPostScriptDriverStr kPMPrinterInfoPrelude "PMIsPostScriptDriver"
#define kPMPrinterIsPostScriptDriverKey CFSTR(kPMPrinterIsPostScriptDriverStr)
```

Constants

`kPMPrinterIsPostScriptDriverStr`
The PostScript printer driver string.

`kPMPrinterIsPostScriptDriverKey`
The value of this key is a `CFBoolean` value.

Print Settings Ticket Keys

Defines strings and keys for items in a print settings ticket.

```
#define kPMBorderStr kPMPrintSettingsPrelude "PMBorder"
#define kPMBorderKey CFSTR(kPMBorderStr)
#define kPMBorderTypeStr kPMPrintSettingsPrelude "PMBorderType"
#define kPMBorderTypeKey CFSTR(kPMBorderTypeStr)
#define kPMLayoutNUpStr kPMPrintSettingsPrelude "PMLayoutNUp"
#define kPMLayoutNUpKey CFSTR(kPMLayoutNUpStr
#define kPMLayoutRowsStr kPMPrintSettingsPrelude "PMLayoutRows"
#define kPMLayoutRowsKey CFSTR(kPMLayoutRowsStr)
#define kPMLayoutColumnsStr kPMPrintSettingsPrelude "PMLayoutColumns"
#define kPMLayoutColumnsKey CFSTR(kPMLayoutColumnsStr)
#define kPMLayoutDirectionStr kPMPrintSettingsPrelude "PMLayoutDirection"
#define kPMLayoutDirectionKey CFSTR(kPMLayoutDirectionStr)
#define kPMLayoutTileOrientationStr kPMPrintSettingsPrelude "PMLayoutTileOrientation"
#define kPMLayoutTileOrientationKey CFSTR(kPMLayoutTileOrientationStr)
#define kPMQualityStr kPMPrintSettingsPrelude "PMQuality"
#define kPMQualityKey CFSTR(kPMQualityStr)
#define kPMPaperTypeStr kPMPrintSettingsPrelude "PMPaperType"
#define kPMPaperTypeKey CFSTR(kPMPaperTypeStr)
#define kPMJobStateStr kPMPrintSettingsPrelude "PMJobState"
#define kPMJobStateKey CFSTR(kPMJobStateStr)
#define kPMJobHoldUntilTimeStr kPMPrintSettingsPrelude "PMJobHoldUntilTime"
#define kPMJobHoldUntilTimeKey CFSTR(kPMJobHoldUntilTimeStr)
#define kPMJobPriorityStr kPMPrintSettingsPrelude "PMJobPriority"
#define kPMJobPriorityKey CFSTR(kPMJobPriorityStr)
#define kPMPaperSourceStr kPMPrintSettingsPrelude "PMPaperSource"
#define kPMPaperSourceKey CFSTR(kPMPaperSourceStr)
#define kPMDuplexingStr kPMPrintSettingsPrelude "PMDuplexing"
#define kPMDuplexingKey CFSTR(kPMDuplexingStr)
#define kPMColorModeStr kPMPrintSettingsPrelude "PMColorMode"
#define kPMColorModeKey CFSTR(kPMColorModeStr)
#define kPMColorSyncProfileIDStr kPMPrintSettingsPrelude "PMColorSyncProfileID"
#define kPMColorSyncProfileIDKey            CFSTR(kPMColorSyncProfileIDStr)
#define kPMColorSyncSystemProfilePathStr    kPMPrintSettingsPrelude
"PMColorSyncSystemProfilePath"
#define kPMColorSyncSystemProfilePathKey CFSTR(kPMColorSyncSystemProfilePathStr)
#define kPMPrintScalingHorizontalStr kPMPrintSettingsPrelude "PMScaling"
#define kPMPrintScalingHorizontalKey CFSTR(kPMPrintScalingHorizontalStr)
#define kPMPrintScalingVerticalStr kPMPrintSettingsPrelude "PMVerticalScaling"
#define kPMPrintScalingVerticalKey CFSTR(kPMPrintScalingVerticalStr)
#define kPMPrintScalingAlignmentStr kPMPrintSettingsPrelude "PMScalingAlignment"
#define kPMPrintScalingAlignmentKey CFSTR(kPMPrintScalingAlignmentStr)
#define kPMPrintOrientationStr kPMPrintSettingsPrelude "PMOrientation"
#define kPMPrintOrientationKey CFSTR(kPMPrintOrientationStr)
#define kPMPreviewStr kPMPrintSettingsPrelude "PMPreview"
#define kPMPreviewKey CFSTR(kPMPreviewStr
#define kPMPrintBackupRecordHdlStr kPMPrintSettingsPrelude "BackupPrintRecordHandle"
#define kPMPrintBackupRecordHdlKey CFSTR(kPMPrintBackupRecordHdlStr)
#define kPMPrintBackupRecordDataStr kPMPrintSettingsPrelude "BackupPrintRecord"
#define kPMPrintBackupRecordDataKey CFSTR(kPMPrintBackupRecordDataStr)
#define kPMPrintCustomDialogHdlStr kPMPrintSettingsPrelude "CustomDialogRecord"
#define kPMPrintCustomDialogHdlKey CFSTR(kPMPrintCustomDialogHdlStr)
#define kPMPrimaryPaperFeedStr                kPMPrintSettingsPrelude
"PMPrimaryPaperFeed"
#define kPMPrimaryPaperFeedKey                CFSTR(kPMPrimaryPaperFeedStr)
#define kPMSecondaryPaperFeedStr            kPMPrintSettingsPrelude
"PMSecondaryPaperFeed"
#define kPMSecondaryPaperFeedKey            CFSTR(kPMSecondaryPaperFeedStr)
#define kPMPSErrorHandlerStr                kPMPrintSettingsPrelude
```

```
"PMPSErrorHandler"
#define kPMPSErrorHandlerKey                CFSTR(kPMPSErrorHandlerStr)
#define kPMPSErrorOnScreenStr                kPMPrintSettingsPrelude
"PMPSErrorOnScreen"
#define kPMPSErrorOnScreenKey                CFSTR(kPMPSErrorOnScreenStr)
#define kPMPSTraySwitchStr                    kPMPrintSettingsPrelude
"PMPSTraySwitch"
#define kPMPSTraySwitchKey                    CFSTR(kPMPSTraySwitchStr)
#define kPMPPDDictStr     kPMPrintSettingsPrelude "kPMPPDDictStr"
#define kPMPPDDictKey CFSTR(kPMPPDDictStr)
```

Constants

`kPMPrintSettingsPrelude`
The print settings ticket prelude string.

`kPMDestinationTypeStr`
The destination type string.

`kPMDestinationTypeKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; can be `kPMDestinationPrinter`, `kPMDestinationFile`, or `kPMDestinationFax`.

`kPMOutputFilenameStr`
The output filename string.

`kPMOutputFilenameKey`
The value of this key is a `CFString`; a URL that specifies the output filename.

`kPMCopiesStr`
The copies string.

`kPMCopiesKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type` that specifies number of copies to print.

`kPMCopyCollateStr`
The copy/collate string.

`kPMCopyCollateKey`
The value of this key is a `CFBoolean` value; used to turn collating on.

`kPMReverseOrderStr`
The reverse order string.

`kPMReverseOrderKey`
The value of this key is a `CFBoolean` value; `true` specifies to print sheets back to front. All layout options are unaffected by reverse order.

`kPMPageRangeStr`
The page range string.

`kPMPageRangeKey`
The value of this key is a `CFArray` of type `kCFNumberSInt32Type`; indicates valid range of pages that the application is able to print.

`kPMFirstPageStr`
The first page string.

`kPMFirstPageKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; the first page selected by user to print.

`kPMLastPageStr`
The last page string.

`kPMLastPageKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; the last page selected by user to print.

`kPMBorderStr`
The border string.

`kPMBorderKey`
The value of this key is a `CFBoolean`; `true` specifies to use borders.

`kPMBorderTypeStr`
The border type string.

`kPMBorderTypeKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration (`PMBorderType`).

`kPMLayoutNUpStr`
The layout string.

`kPMLayoutNUpKey`
The value of this key is a `CFBoolean` value; turns N-Up layout on.

`kPMLayoutRowsStr`
The layout rows string.

`kPMLayoutRowsKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; indicates number of layout rows.

`kPMLayoutColumnsStr`
The layout columns string.

`kPMLayoutColumnsKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; indicates number of layout columns.

`kPMLayoutDirectionStr`
The layout direction string.

`kPMLayoutDirectionKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration (`PMLayoutDirection`).

`kPMLayoutTileOrientationStr`
The layout tile orientation string.

`kPMLayoutTileOrientationKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; `PMOrientation`, 1 = portrait, 2 = landscape, etc.

`kPMQualityStr`
The print quality string.

`kPMQualityKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specfies an enumeration—draft, normal, best.

`kPMPaperTypeStr`
The paper type string.

`kPMPaperTypeKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`.

`kPMJobStateStr`
The job state string.

`kPMJobStateKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration—active = 0, pending, hold until, hold indefinitely, aborted, finished.

`kPMJobHoldUntilTimeStr`
The job hold until string.

`kPMJobHoldUntilTimeKey`
The value of this key is a `CFDate`; specifies time to print the job.

`kPMJobPriorityStr`
The job priority string.

`kPMJobPriorityKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration—Low = 0, normal, urgent.

`kPMPaperSourceStr`
The paper source string.

`kPMPaperSourceKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration of paper sources.

`kPMDuplexingStr`
The duplexing string.

`kPMDuplexingKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration—`kPMDuplexNone`, `kPMDuplexNoTumble`, `kPMDuplexTumble`, `kPMSimplexTumble`.

`kPMColorModeStr`
The color mode string.

`kPMColorModeKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration—Black and White, Grayscale, Color, HiFi Color.

`kPMColorSyncProfileIDStr`
The ColorSync Profile ID string.

`kPMColorSyncProfileIDKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies the ID of the ColorSync profile to use.

`kPMColorSyncSystemProfilePathStr`
The ColorSync system profile path string.

`kPMColorSyncSystemProfilePathKey`
The value of this key is a `CFString`; specifies the path of system profile.

`kPMPrintScalingHorizontalStr`
The horizontal print scaling string.

`kPMPrintScalingHorizontalKey`
The value of this key is a `CFNumber` of type `kCFNumberDoubleType`; specifies the horizontal scaling factor applied to original page size.

`kPMPrintScalingVerticalStr`
The vertical print scaling string.

`kPMPrintScalingVerticalKey`
The value of this key is a `CFNumber` of type `kCFNumberDoubleType`; specifies the vertical scaling factor applied to original page size.

`kPMPrintScalingAlignmentStr`
The print scaling alignment string.

`kPMPrintScalingAlignmentKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies an enumeration (`PMScalingAlignment`).

`kPMPrintOrientationStr`
The print orientation string.

`kPMPrintOrientationKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`; specifies and enumeration—`PMOrientation`, 1 = portrait, 2 = landscape, etc.

`kPMPreviewStr`
The print preview string.

`kPMPreviewKey`
The value of this key is a `CFString`; "YES" indicates the user clicked the Preview button.

`kPMPrintBackupRecordHdlStr`
The print backup record handle string.

`kPMPrintBackupRecordHdlKey`
The value of this key is a print record handle (`CFData`); not used when data is flattened. (Mac OS 8 and 9 only)

`kPMPrintBackupRecordDataStr`
The print backup record data string.

`kPMPrintBackupRecordDataKey`
The value of this key is a print record (`CFData`) stored in complete form; used when flattening a ticket with the print record. (Mac OS 8 and 9 only)

`kPMPrintCustomDialogHdlStr`
The print custom dialog handle string.

`kPMPrintCustomDialogHdlKey`
The value of this key is a handle (`CFData`) to the print record using for custom dialog calls; not stored when data is flattened. (Mac OS 8 and 9 only)

`kPMPrimaryPaperFeedStr`
The primary paper feed string.

`kPMPrimaryPaperFeedKey`
The value of this key is a `CFArray`; the main and option PPD keywords for input paper feed.

`kPMSecondaryPaperFeedStr`
The secondary paper feed string.

`kPMSecondaryPaperFeedKey`
The value of this key is a `CFArray`; the main and option PPD keywords for input paper feed.

`kPMPSErrorHandlerStr`
The PostScript error handle string.

`kPMPSErrorHandlerKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type`.

`kPMPSErrorOnScreenStr`
The PostScript error on screen string.

`kPMPSErrorOnScreenKey`
The value of this key is a `CFBoolean`; turns PostScript error onscreen notification on.

`kPMPSTraySwitchStr`
The PostScript tray switch string.

`kPMPSTraySwitchKey`
The value of this key is a `CFArray`; the main and option PostScript printer description file keywords for tray switching.

`kPMPPDDictStr`
The PostScript printer description file string.

`kPMPPDDictKey`
The value of this key is a `CFDictionary`; the main and option PostScript printer description file keywords for additional features.

Printer Driver Creator Code Key

Defines a string and key for the printer driver creator code.

```
#define kPMDriverCreatorStr                    kPMPrinterInfoPrelude
"PMDriverCreator        "
#define kPMDriverCreatorKey                    CFSTR(kPMDriverCreatorStr)
```

Constants

`kPMDriverCreatorStr`
The printer driver creator string.

`kPMDriverCreatorKey`
The value of this key is a `CFNumber` of type `kCFNumberSInt32Type` that specifies the creator code for the printer driver.

Printer Font Keys

Defines a string and key for the printer fonts.

```
#define kPMPrinterFontStr kPMPrinterInfoPrelude "Printer
Fonts    "
#define kPMPrinterFontKey CFSTR(kPMPrinterFontStr)
```

Constants

`kPMPrinterFontStr`
The printer font string.

`kPMPrinterFontKey`
The value of this key is `CFData` that specifies the printer resident fonts.

Printer Info Ticket Keys

Defines strings and keys for items in a printer info ticket.

```
#define kPMPrinterInfoPrelude "com.apple.print.PrinterInfo."
#define kPMPrinterLongNameStr kPMPrinterInfoPrelude "PMPrinterLongName"
#define kPMPrinterLongNameKey CFSTR(kPMPrinterLongNameStr)
#define kPMPrinterShortNameStr kPMPrinterInfoPrelude "PMPrinterShortName"
#define kPMPrinterShortNameKey CFSTR(kPMPrinterShortNameStr)
#define kPMMakeAndModelNameStr kPMPrinterInfoPrelude "PMMakeAndModelName"
#define kPMMakeAndModelNameKey CFSTR(kPMMakeAndModelNameStr)
#define kPMPrinterAddressStr kPMPrinterInfoPrelude "PMPrinterAddress"
#define kPMPrinterAddressKey CFSTR(kPMPrinterAddressStr)
#define kPMSupportsColorStr kPMPrinterInfoPrelude "PMSupportsColor"
#define kPMSupportsColorKey CFSTR(kPMSupportsColorStr)
#define kPMDoesCopiesStr kPMPrinterInfoPrelude "PMDoesCopies"
#define kPMDoesCopiesKey CFSTR(kPMDoesCopiesStr)
#define kPMDoesCopyCollateStr kPMPrinterInfoPrelude "PMDoesCopyCollate"
#define kPMDoesCopyCollateKey CFSTR(kPMDoesCopyCollateStr)
#define kPMDoesReverseOrderStr kPMPrinterInfoPrelude "PMDoesReverseOrderK"
#define kPMDoesReverseOrderKey CFSTR(kPMDoesReverseOrderStr)
#define kPMInputFileTypeListStr kPMPrinterInfoPrelude "PMInputFileTypeList"
#define kPMInputFileTypeListKey CFSTR(kPMInputFileTypeListStr)
#define kPMOutputTypeListStr     kPMPrinterInfoPrelude "PMOutputTypeList"
#define kPMOutputTypeListKey     CFSTR(kPMOutputTypeListStr)
```

Constants

`kPMPrinterInfoPrelude`
The printer info ticket prelude string.

`kPMPrinterLongNameStr`
The printer long name string.

`kPMPrinterLongNameKey`
The value of this key is a `CFString`; specifies the full name of the printer.

`kPMPrinterShortNameStr`
The printer short name string.

`kPMPrinterShortNameKey`
The value of this key is a `CFString`; specifies a shortened version of the printer name.

`kPMMakeAndModelNameStr`
The printer make and model string.

`kPMMakeAndModelNameKey`
The value of this key is a `CFString`; specifies the product name used for the printer

`kPMPrinterAddressStr`
The printer address string.

`kPMPrinterAddressKey`
The value of this key is the product address (`CFData`).

`kPMSupportsColorStr`
The color supported string.

`kPMSupportsColorKey`
The value of this key is a `CFBoolean`; `true` if the printer supports color printing.

`kPMDoesCopiesStr`
The copies string.

`kPMDoesCopiesKey`
The value of this key is a `CFBoolean`; `true` if the printer supports copies.

`kPMDoesCopyCollateStr`
The collation string.

`kPMDoesCopyCollateKey`
The value of this key is a `CFBoolean`; `true` if the printer supports collation.

`kPMDoesReverseOrderStr`
The reverse-order string.

`kPMDoesReverseOrderKey`
The value of this key is a `CFBoolean`; `true` if the printer can print in reverse order.

`kPMInputFileTypeListStr`
The file type list string.

`kPMInputFileTypeListKey`
The value of this key is a `CFArray` of `CFString` data that indicates file types.

`kPMOutputTypeListStr`
The output type list string.

`kPMOutputTypeListKey`
The value of this key is a `CFArray` of `CFString` data that indicates the MIME types for the data that can be sent to an I/O module.

Printing Phase Types

Specify the phase of the printing process.

```
typedef UInt16 PMPrintingPhaseType;
enum {
    kPMPhaseUnknown = 0,
    kPMPhasePreDialog = 1,
    kPMPhaseDialogsUp = 2,
    kPMPhasePostDialogs = 3,
    kPMPhasePreAppDrawing = 4,
    kPMPhaseAppDrawing = 5,
    kPMPhasePostAppDrawing = 6,
    kPMPhasePreConversion = 7,
    kPMPhaseConverting = 8,
    kPMPhasePostConversion = 9,
    kPMPhasePrinting = 10
};
```

Constants

`kPMPhaseUnknown`
Phase unknown.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhasePreDialog`
Just before the code to open the dialog.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhaseDialogsUp`
A printing dialogs is open.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhasePostDialogs`
The printing dialogs have been opened and are now closed, but the job is not yet spooling.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhasePreAppDrawing`
The job is just about to spool.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhaseAppDrawing`
Drawing commands are now being spooled from the application.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhasePostAppDrawing`
Spooling is finished, not the job is not yet rendered or converted.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhasePreConversion`
The job is just about to be converted to its final format (PostScript, Raster, or other).

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhaseConverting`
The job is being converted from the spool file to the final format.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhasePostConversion`
The data is ready for the printer and waiting for completion.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

`kPMPhasePrinting`
The job is waiting for the printer.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `PMTicket.h`.

Discussion
Printing phase types signal a shift from one printing phase to the next and are set by many parts of the printing system. You can check the phase by testing for greater-than and less-than conditions.

Rasterizer Options

Specify options for rasterizing.

```
enum {
    kPMPSTTRasterizerUnknown = 0,
    kPMPSTTRasterizerNone = 1,
    kPMPSTTRasterizerAccept68K = 2,
    kPMPSTTRasterizerType42 = 3
};
```

Constants

`kPMPSTTRasterizerUnknown`
Rasterizer unknown.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPSTTRasterizerNone`
No rasterizer.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPSTTRasterizerAccept68K`
Accepts 68 K.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPSTTRasterizerType42`
Uses type 42.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

Template Entry Data Types

Specify the data type of a job template entry.

```
typedef SInt32 PMValueType;
enum {
    kPMValueUndefined = 0,
    kPMValueBoolean = 1,
    kPMValueData = 2,
    kPMValueString = 3,
    kPMValueSInt32 = 4,
    kPMValueSInt32Range = 5,
    kPMValueUInt32 = 6,
    kPMValueUInt32Range = 7,
    kPMValueDouble = 8,
    kPMValueDoubleRange = 9,
    kPMValuePMRect = 10,
    kPMValueDate = 11,
    kPMValueArray = 12,
    kPMValueDict = 13,
    kPMValueTicket = 14
};
```

Constants

`kPMValueUndefined`
Template entry is unknown or undefined.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueBoolean`
A `CFBoolean` value.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueData`
A `CFData` value. This is generic data converted to Core Foundation data. Template entries of this type should have a default value, but no other constraints.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueString`
A `CFString` value. Template entries of this type should have a default value, but no other constraints.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueSInt32`
A `CFNumber` value.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueSInt32Range`
A pair of `CFNumber` values (`SInt32` data type).

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueUInt32`
A `CFNumber` value (unsigned long data types).

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueUInt32Range`
A pair of `CFNumber` values (`UInt32` values).

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueDouble`
A `CFNumber` value (`double` data type.)

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueDoubleRange`
A pair of `CFNumber` values (`double` data types).

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValuePMRect`
A Core Foundation array that contains four `CFNumbers` values (`double` data types).

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueDate`
A `CFDate` value.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueArray`
A Core Foundation array.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueDict`
A `CFDictionary` data structure. Template entries of this type should have a default value, but no other constraints.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

`kPMValueTicket`
A ticket.

Available in Mac OS X v10.0 and later.

Declared in `PMTemplate.h`.

Discussion
A template entry data type determines what other fields and functions are available for the entry.

Template Strings

Define a string associated with a template.

```
#define kPMTemplatePrelude     "com.apple.print.TemplateSpecific."
#define kPMPaperInfoListStr     "PMTemplatePaperInfoTicket"
#define kPMPaperInfoList CFSTR( kPMPaperInfoListStr )
```

Constants

`kPMTemplatePrelude`
A template prelude string.

`kPMPaperInfoListStr`
A paper info list string.

`kPMPaperInfoList`
A paper info list.

Ticket Levels

Specify the level of an item within a ticket.

```
enum {
    kPMTopLevel = 0
};
```

Constants

`kPMTopLevel`
Specifies the top level ticket.

Discussion

You should pass the constant `kPMTopLevel` to any function that has parameters to specify ticket level calls. No other options are currently available.

Ticket Types

Specify a ticket type, that is, a ticket kind.

```
typedef SInt16 PMTicketType;
enum {
    kPMTicketTypeUnknown = -1,
    kPMJobTicketType = 1,
    kPMDocumentTicketType = 2,
    kPMPageTicketType = 3,
    kPMPageFormatTicketType = 4,
    kPMPrintSettingsTicketType = 5,
    kPMPrinterInfoTicketType = 6,
    kPMDestinationTicketType = 7,
    kPMConverterSetupTicketType = 8,
    kPMModuleInfoTicketType = 9,
    kPMTicketListType = 10,
    kPMPaperInfoTicketType = 11
};
```

Constants

`kPMTicketTypeUnknown`
Specifies the ticket type is not know.

Available in Mac OS X v10.0 and later.

Declared in `PMTicket.h`.

`kPMJobTicketType`
Specifies a job ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMDocumentTicketType`
Specifies a document ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPageTicketType`
Specifies a page ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPageFormatTicketType`
Specifies a page format ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPrintSettingsTicketType`
Specifies a print settings ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPrinterInfoTicketType`
Specifies a printer info ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMDestinationTicketType`
Specifies a destination ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMConverterSetupTicketType`
Specifies a converter setup ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMModuleInfoTicketType`
Specifies a printer module info ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMTicketListType`
Specifies a list ticket.
Available in Mac OS X v10.0 and later.
Declared in `PMTicket.h`.

`kPMPaperInfoTicketType`
Specifies a paper info ticket.

Available in Mac OS X v10.0 and later.

Declared in `PMTicket.h`.

Ticket Type Strings

Specify ticket types.

```
#define kPMJobTicket CFSTR("com.apple.print.JobTicket")
#define kPMDocumentTicket CFSTR("com.apple.print.DocumentTicket")
#define kPMPageTicket CFSTR("com.apple.print.PageTicket")
#define kPMPageFormatTicket CFSTR("com.apple.print.PageFormatTicket")
#define kPMPrintSettingsTicket CFSTR("com.apple.print.PrintSettingsTicket")
#define kPMDestinationTicket CFSTR("com.apple.print.DestinationTicket")
#define kPMConverterSetupTicket CFSTR("com.apple.print.ConverterSetupTicket")
#define kPMPrinterInfoTicket CFSTR("com.apple.print.PrinterInfoTicket")
#define kPMModuleInfoTicket CFSTR("com.apple.print.ModuleInfoTicket")
#define kPMTicketList CFSTR("com.apple.print.TicketList")
#define kPMPaperInfoTicket CFSTR("com.apple.print.PaperInfoTicket")
```

Constants

`kPMJobTicket`
Specifies a job ticket; the top-level ticket for a print job.

`kPMDocumentTicket`
Specifies a document ticket.

`kPMPageTicket`
Specifies a page ticket.

`kPMPageFormatTicket`
Specifies a page format ticket.

`kPMPrintSettingsTicket`
Specifies a print settings ticket.

`kPMDestinationTicket`
Specifies a destination ticket.

`kPMConverterSetupTicket`
Specifies a converter setup ticket.

`kPMPrinterInfoTicket`
Specifies a printer info ticket.

`kPMModuleInfoTicket`
Specifies a module info ticket.

`kPMTicketList`
Specifies a list ticket.

`kPMPaperInfoTicket`
Specifies a paper info ticket.

Discussion

Use these ticket type strings of ticket types to create a ticket.

Result Codes

This table lists result codes defined for Ticket Services.

Result Code	Value	Description
kPMTicketTypeNotFound	-9580	The ticket type was not found. Available in Mac OS X v10.0 and later.
kPMUpdateTicketFailed	-9581	Could not update the ticket. Available in Mac OS X v10.0 and later.
kPMValidateTicketFailed	-9582	Could not validate the ticket. Available in Mac OS X v10.0 and later.
kPMSubTicketNotFound	-9583	Could not find the subticket. Available in Mac OS X v10.0 and later.
kPMInvalidSubTicket	-9584	The subticket is invalid. Available in Mac OS X v10.0 and later.
kPMDeleteSubTicketFailed	-9585	Could not delete the subticket. Available in Mac OS X v10.0 and later.
kPMItemIsLocked	-9586	The item is locked. Available in Mac OS X v10.0 and later.
kPMTicketIsLocked	-9587	The ticket is locked. Available in Mac OS X v10.0 and later.
kPMTemplateIsLocked	-9588	The job template is locked. Available in Mac OS X v10.0 and later.
kPMKeyNotFound	-9589	Could not find the key. Available in Mac OS X v10.0 and later.
kPMKeyNotUnique	-9590	The key is not unique. Available in Mac OS X v10.0 and later.
kPMUnknownDataType	-9591	The data type is unknown. Available in Mac OS X v10.0 and later.

Other References

ATSUI Reference

Framework: ApplicationServices/ApplicationServices.h

Declared in ATSUnicode.h
ATSUnicodeDirectAccess.h
ATSUnicodeDrawing.h
ATSUnicodeFlattening.h
ATSUnicodeFonts.h
ATSUnicodeGlyphs.h
ATSUnicodeObjects.h
ATSUnicodeTypes.h
ATSLayoutTypes.h

Overview

Apple Type Services for Unicode Imaging (ATSUI) enables the rendering of Unicode-encoded text with advanced typographic features. It automatically handles many of the complexities inherent in text layout, including the correct rendering of text in bidirectional and vertical script systems.

ATSUI may be useful to developers who are writing new text editors or word processing applications that render Unicode-encoded text. You can also use ATSUI if you want to modify your existing application to support Unicode text rendering.

This document describes the ATSUI application programming interface (API) through version 2.4. If you are a font designer or want more information about fonts, see the Apple font site: http://developer.apple.com/fonts/

Functions by Task

Creating and Initializing Style Objects

`ATSUCreateStyle` (page 1865)
Creates an opaque style object containing only default style attributes, font features, and font variations.

`ATSUCreateAndCopyStyle` (page 1863)
Creates a copy of a style object.

`ATSUCompareStyles` (page 1855)
Compares the attribute values of two style objects.

`ATSUClearStyle` (page 1854)
Restores default values to a style object.

`ATSUStyleIsEmpty` (page 1968)
Indicates whether a style object contains only default values.

`ATSUSetStyleRefCon` (page 1961)
Sets application-specific data for a style object.

`ATSUGetStyleRefCon` (page 1918)
Obtains application-specific data for a style object.

`ATSUDisposeStyle` (page 1876)
Disposes of the memory associated with a style object.

Manipulating Style Attributes

`ATSUSetAttributes` (page 1950)
Sets style attribute values in a style object.

`ATSUCopyAttributes` (page 1856)
Copies all style attribute settings from a source style object to a destination style object.

`ATSUOverwriteAttributes` (page 1944)
Copies to a destination style object the nondefault style attribute settings of a source style object.

`ATSUUnderwriteAttributes` (page 1972)
Copies to a destination style object only those nondefault style attribute settings of a source style object that are at default settings in the destination object.

`ATSUGetAllAttributes` (page 1886)
Obtains an array of style attribute tags and value sizes for a style object.

`ATSUGetAttribute` (page 1892)
Obtains a style attribute value for a style object.

`ATSUGetContinuousAttributes` (page 1893)
Obtains the style attribute values that are continuous over a given text range.

`ATSUClearAttributes` (page 1848)
Restores default values to the specified style attributes of a style object.

Manipulating Font Features

`ATSUSetFontFeatures` (page 1952)
Sets font features in a style object.

`ATSUGetAllFontFeatures` (page 1887)
Obtains the font features of a style object that are not at default settings.

`ATSUGetFontFeature` (page 1895)
Obtains the font feature corresponding to an index into an array of font features for a style object.

`ATSUClearFontFeatures` (page 1849)
Restores default settings to the specified font features of a style object.

`ATSUGetFontFeatureTypes` (page 1898)
Obtains the available feature types of a font.

`ATSUCountFontFeatureTypes` (page 1860)
Obtains the number of available feature types in a font.

ATSUGetFontFeatureSelectors (page 1897)
Obtains the available feature selectors for a given feature type in a font.

ATSUCountFontFeatureSelectors (page 1859)
Obtains the number of available feature selectors for a given feature type in a font.

Manipulating Font Variations

ATSUSetVariations (page 1967)
Sets font variation axes and values in a style object.

ATSUGetAllFontVariations (page 1888)
Obtains a style object's font variation values that are not at default settings.

ATSUGetFontVariationValue (page 1903)
Obtains the current value for a font variation axis in a style object.

ATSUClearFontVariations (page 1850)
Restores default values to the specified font variation axes of a style object.

ATSUGetIndFontVariation (page 1911)
Obtains a variation axis and its value range for a font.

ATSUCountFontVariations (page 1862)
Obtains the number of defined variation axes in a font.

ATSUGetFontInstance (page 1900)
Obtains the font variation axis values for a font instance.

ATSUCountFontInstances (page 1860)
Obtains the number of defined font instances in a font.

Creating and Initializing Text Layout Objects

ATSUCreateTextLayout (page 1866)
Creates an opaque text layout object containing only default text layout attributes.

ATSUCreateTextLayoutWithTextPtr (page 1869)
Creates an opaque text layout object containing default text layout attributes as well as associated text and text styles.

ATSUCreateAndCopyTextLayout (page 1863)
Creates a copy of a text layout object.

ATSUSetTextPointerLocation (page 1965)
Associates text with a text layout object or updates previously associated text.

ATSUGetTextLocation (page 1921)
Obtains information about the text associated with a text layout object.

ATSUSetRunStyle (page 1959)
Defines a style run by associating style information with a run of text.

ATSUGetRunStyle (page 1916)
Obtains style run information for a character offset in a run of text.

ATSUSetTextLayoutRefCon (page 1964)
Sets application-specific data for a text layout object.

`ATSUGetTextLayoutRefCon` (page 1921)
Obtains application-specific data for a text layout object.

`ATSUDisposeTextLayout` (page 1876)
Disposes of the memory associated with a text layout object.

Manipulating Text Layout Attributes

`ATSUSetLayoutControls` (page 1955)
Sets layout control attribute values in a text layout object.

`ATSUCopyLayoutControls` (page 1857)
Copies all layout control attribute settings from a source text layout object to a destination text layout object.

`ATSUGetAllLayoutControls` (page 1890)
Obtains an array of layout control attribute tags and value sizes for a text layout object.

`ATSUGetLayoutControl` (page 1912)
Obtains a layout control attribute value for a text layout object.

`ATSUClearLayoutControls` (page 1852)
Restores default values to the specified layout control attributes of a text layout object.

Manipulating Line Attributes

`ATSUSetLineControls` (page 1956)
Sets layout control attribute values for a single line in a text layout object.

`ATSUCopyLineControls` (page 1857)
Copies line control attribute settings from a line in a source text layout object to a line in a destination text layout object.

`ATSUGetAllLineControls` (page 1891)
Obtains an array of line control attribute tags and value sizes for a line in a text layout object.

`ATSUGetLineControl` (page 1913)
Obtains a line control attribute value for a line in a text layout object.

`ATSUClearLineControls` (page 1853)
Restores default values to the specified line control attributes of a line in a text layout object.

Manipulating Line Breaks

`ATSUBreakLine` (page 1846)
Calculates and, optionally, sets a soft line break in a range of text.

`ATSUBatchBreakLines` (page 1844)
Calculates soft line breaks for the text associated with a text layout object.

`ATSUSetSoftLineBreak` (page 1961)
Sets a soft line break that you specify.

`ATSUGetSoftLineBreaks` (page 1917)
Obtains soft line breaks in a range of text.

ATSUClearSoftLineBreaks (page 1854)
Removes soft line breaks from a range of text.

Substituting Fonts

ATSUMatchFontsToText (page 1936)
Examines a text range for characters that cannot be drawn with the current font and suggests a substitute font, if necessary.

ATSUSetTransientFontMatching (page 1967)
Turns automatic font substitution on or off for a text layout object.

ATSUGetTransientFontMatching (page 1922)
Obtains whether ATSUI automatically performs font substitution for a text layout object.

ATSUCreateFontFallbacks (page 1864)
Creates an opaque object that can be set to contain a font list and a font-search method.

ATSUSetObjFontFallbacks (page 1958)
Assigns a font list and a font-search method to a font fallback object.

ATSUGetObjFontFallbacks (page 1914)
Obtains the font list and font-search method associated with a font fallback object.

ATSUDisposeFontFallbacks (page 1875)
Disposes of the memory associated with a font fallback object.

Identifying Fonts

ATSUGetFontIDs (page 1899)
Obtains a list of all the ATSUI-compatible fonts installed on the user's system.

ATSUFontCount (page 1885)
Obtains the number of ATSUI-compatible fonts installed on a user's system.

ATSUFindFontName (page 1880)
Obtains a name string and index value for the first font in a name table that matches the specified ATSUI font ID, name code, platform, script, and/or language.

ATSUFindFontFromName (page 1879)
Obtains an ATSUI font ID for the first entry in a name table that matches the specified name string, name code, platform, script, and/or language.

ATSUGetIndFontName (page 1908)
Obtains a name string, name code, platform, script, and language for the font that matches an ATSUI font ID and name table index value.

ATSUCountFontNames (page 1861)
Obtains the number of font names that correspond to a given ATSUI font ID.

ATSUGetIndFontTracking (page 1910)
Obtains the name code and tracking value for the font tracking that matches an ASTUI font ID, glyph orientation, and tracking table index.

ATSUCountFontTracking (page 1861)
Obtains the number of entries in the font tracking table that correspond to a given ATSUI font ID and glyph orientation.

`ATSUGetFontFeatureNameCode` (page 1896)
Obtains the name code for a font's feature type or selector that matches an ASTUI font ID, feature type, and feature selector.

`ATSUGetFontVariationNameCode` (page 1902)
Obtains the name code for the font variation that matches an ASTUI font ID and font variation axis.

`ATSUGetFontInstanceNameCode` (page 1901)
Obtains the name code for the font instance that matches an ASTUI font ID and font instance index value.

Drawing and Highlighting Text

`ATSUDrawText` (page 1877)
Renders a range of text at a specified location in a QuickDraw graphics port or Quartz graphics context.

`ATSUHighlightText` (page 1931)
Renders a highlighted range of text at a specified location in a QuickDraw graphics port or Quartz graphics context.

`ATSUUnhighlightText` (page 1974)
Renders a previously highlighted range of text in an unhighlighted state.

`ATSUSetHighlightingMethod` (page 1953)
Sets the method ATSUI uses to highlight and unhighlight text for a text layout object.

`ATSUGetTextHighlight` (page 1919)
Obtains the highlight region for a range of text.

`ATSUHighlightInactiveText` (page 1930)
Highlights previously selected text using an alpha value of 0.5.

`ATSUClearLayoutCache` (page 1851)
Clears the layout cache of a line or an entire text layout object.

Supporting User Interaction With Onscreen Text

`ATSUTextInserted` (page 1970)
Informs ATSUI of the location and length of a text insertion.

`ATSUTextDeleted` (page 1969)
Informs ATSUI of the location and length of a text deletion.

`ATSUTextMoved` (page 1971)
Informs ATSUI of the new memory location of relocated text.

`ATSUPositionToOffset` (page 1946)
Obtains the memory offset for the glyph edge nearest a mouse-down event.

`ATSUOffsetToPosition` (page 1942)
Obtains the caret position(s) corresponding to a memory offset.

`ATSUNextCursorPosition` (page 1940)
Obtains the memory offset for the insertion point that follows the current insertion point in storage order, as determined by a move of the specified length.

ATSUPreviousCursorPosition (page 1948)
Obtains the memory offset for the insertion point that precedes the current insertion point in storage order, as determined by a move of the specified length.

ATSURightwardCursorPosition (page 1949)
Obtains the memory offset for the insertion point to the right of the high caret position, as determined by a move of the specified length at a line direction boundary.

ATSULeftwardCursorPosition (page 1933)
Obtains the memory offset for the insertion point to the left of the high caret position, as determined by a move of the specified length at a line direction boundary.

ATSUPositionToCursorOffset (page 1945)
Obtains the memory offset for the glyph edge nearest a mouse-down event, after a move of the specified length.

ATSUOffsetToCursorPosition (page 1941)
Obtains the caret position(s) corresponding to a memory offset, after a move of the specified length.

Obtaining Text Metrics

ATSUMeasureTextImage (page 1938)
Obtains the image bounding rectangle for a line of text after final layout.

ATSUGetUnjustifiedBounds (page 1923)
Obtains the typographic bounding rectangle for a line of text prior to final layout.

ATSUGetGlyphBounds (page 1904)
Obtains the typographic bounds of a line of glyphs after final layout.

ATSUCalculateBaselineDeltas (page 1847)
Obtains the optimal baseline positions for glyphs in a style run.

ATSUGlyphGetIdealMetrics (page 1927)
Obtains resolution-independent font metric information for glyphs associated with a given style object.

ATSUGlyphGetScreenMetrics (page 1929)
Obtains device-adjusted font metric information for glyphs associated with a given style object.

ATSUGetNativeCurveType (page 1914)
Obtains the type of outline path used for glyphs associated with a given style object.

ATSUGlyphGetCurvePaths (page 1926)
Obtains the outline paths for a glyph associated with a given style object.

ATSUGlyphGetCubicPaths (page 1925)
Obtains the cubic outline paths for a glyph.

ATSUGlyphGetQuadraticPaths (page 1928)
Obtains the quadratic outline paths for a glyph.

Working With Tabs

ATSUSetTabArray (page 1962)
Sets a tab ruler for a text layout object.

ATSUGetTabArray (page 1918)
Retrieves the tab ruler associated with a text layout object.

Accessing Glyph Data

ATSUDirectGetLayoutDataArrayPtrFromLineRef (page 1872)
Obtains the glyph data specified by a direct-data selector and for a specific line of text.

ATSUDirectGetLayoutDataArrayPtrFromTextLayout (page 1873)
Obtains a copy of the glyph data specified by a direct-data selector and for a specific line of text in a text layout object.

ATSUDirectReleaseLayoutDataArrayPtr (page 1874)
Releases a pointer to a direct-data array.

ATSUDirectAddStyleSettingRef (page 1871)
Looks up, and if necessary, adds a style setting to a line of text.

Flattening and Parsing Style Data

ATSUFlattenStyleRunsToStream (page 1882)
Flattens ATSUI style-run data so that it can be saved to disk or passed (through the pasteboard) to another application.

ATSUUnflattenStyleRunsFromStream (page 1972)
Unflattens previously-flattened ATSUI style run data so that it can be read from disk or accepted (through the pasteboard) from another application.

Creating, Calling, and Deleting Universal Procedure Pointers

NewATSUDirectLayoutOperationOverrideUPP (page 1989)
Creates a new universal procedure pointer (UPP) to a layout operation override callback.

InvokeATSUDirectLayoutOperationOverrideUPP (page 1984)
Calls your layout operation override callback.

DisposeATSUDirectLayoutOperationOverrideUPP (page 1979)
Disposes of a universal procedure pointer (UPP) to a layout operation override callback.

NewRedrawBackgroundUPP (page 1989)
Creates a new universal procedure pointer (UPP) to a redraw background callback.

InvokeRedrawBackgroundUPP (page 1984)
Invokes your redraw background callback.

DisposeRedrawBackgroundUPP (page 1980)
Disposes of a new universal procedure pointer (UPP) to a redraw background callback.

NewATSCubicMoveToUPP (page 1986)
Creates a new universal procedure pointer (UPP) to a cubic move-to callback.

InvokeATSCubicMoveToUPP (page 1981)
Calls your cubic move-to callback.

`DisposeATSCubicMoveToUPP` (page 1977)
Disposes of a universal procedure pointer (UPP) to a cubic move-to callback.

`NewATSCubicLineToUPP` (page 1986)
Creates a new universal procedure pointer (UPP) to a cubic line-to callback.

`InvokeATSCubicLineToUPP` (page 1981)
Calls your cubic line-to callback.

`DisposeATSCubicLineToUPP` (page 1977)
Disposes of a universal procedure pointer (UPP) to a cubic line-to callback.

`NewATSCubicCurveToUPP` (page 1985)
Creates a new universal procedure pointer (UPP) to a cubic curve-to callback.

`InvokeATSCubicCurveToUPP` (page 1980)
Calls your cubic curve-to callback.

`DisposeATSCubicCurveToUPP` (page 1976)
Disposes of a universal procedure pointer (UPP) to a cubic curve-to callback.

`NewATSCubicClosePathUPP` (page 1985)
Creates a new universal procedure pointer (UPP) to a cubic close-path callback.

`InvokeATSCubicClosePathUPP` (page 1980)
Calls your cubic close-path callback.

`DisposeATSCubicClosePathUPP` (page 1976)
Disposes of a universal procedure pointer (UPP) to a cubic close-path callback.

`NewATSQuadraticNewPathUPP` (page 1988)
Creates a new universal procedure pointer (UPP) to a quadratic new-path callback.

`InvokeATSQuadraticNewPathUPP` (page 1983)
Calls your quadratic new-path callback.

`DisposeATSQuadraticNewPathUPP` (page 1979)
Disposes of a universal procedure pointer (UPP) to a quadratic new-path callback.

`NewATSQuadraticLineUPP` (page 1988)
Creates a new universal procedure pointer (UPP) to a quadratic line callback.

`InvokeATSQuadraticLineUPP` (page 1983)
Calls your quadratic line callback.

`DisposeATSQuadraticLineUPP` (page 1978)
Disposes of a universal procedure pointer (UPP) to a quadratic line callback.

`NewATSQuadraticCurveUPP` (page 1987)
Creates a new universal procedure pointer (UPP) to a quadratic curve callback.

`InvokeATSQuadraticCurveUPP` (page 1982)
Calls your quadratic curve callback.

`DisposeATSQuadraticCurveUPP` (page 1978)
Disposes of a universal procedure pointer (UPP) to a quadratic curve callback.

`NewATSQuadraticClosePathUPP` (page 1987)
Creates a new universal procedure pointer (UPP) to a quadratic close-path callback.

`InvokeATSQuadraticClosePathUPP` (page 1982)
Calls your quadratic close-path callback.

`DisposeATSQuadraticClosePathUPP` (page 1978)
Disposes of a universal procedure pointer (UPP) to a quadratic close-path callback.

Not Recommended

`ATSUFONDtoFontID` (page 1884)
Finds the ATSUI font ID that corresponds to a font family number, if one exists. (Deprecated. There is no replacement because FONDs are a QuickDraw concept and QuickDraw is deprecated.)

`ATSUFontIDtoFOND` (page 1885)
Finds the font family number that corresponds to an ATSUI font ID, if one exists. (Deprecated. There is no replacement because FONDs are a QuickDraw concept and QuickDraw is deprecated.)

`ATSUMeasureText` (page 1938)
(Deprecated. Use `ATSUGetUnjustifiedBounds` (page 1923) instead.)

`ATSUDrawGlyphInfo` (page 1877) Deprecated in Mac OS X v10.3
Draws glyphs at the specified location, based on style and layout information specified for each glyph. (Deprecated. Use functions from "Accessing Glyph Data" (page 1842) instead.)

`ATSUGetFontFallbacks` (page 1894) Deprecated in Mac OS X v10.3
Obtains the global font list and search order that ATSUI uses when a font does not have the glyph needed to image a character. (Deprecated. Use font fallback objects instead.)

`ATSUGetGlyphInfo` (page 1906) Deprecated in Mac OS X v10.3
Obtains a copy of the style and layout information for each glyph in a line. (Deprecated. Use functions from "Accessing Glyph Data" (page 1842) instead.)

`ATSUSetFontFallbacks` (page 1952) Deprecated in Mac OS X v10.3
Sets, on a global scope, the font list and search order for ATSUI to use when a font does not have the glyph needed to image a character. (Deprecated. Use font fallback objects instead.)

`ATSUCopyToHandle` (page 1858) Deprecated in Mac OS X v10.1
Copies an ATSUI style to a handle. (Deprecated. Use `ATSUFlattenStyleRunsToStream` (page 1882) instead.)

`ATSUCreateTextLayoutWithTextHandle` (page 1867) Deprecated in Mac OS X v10.0
Creates an opaque text layout object containing default text layout attributes as well as associated text and text styles. (Deprecated. Use `ATSUCreateTextLayoutWithTextPtr` (page 1869) instead. See the Discussion for more details.)

`ATSUIdle` (page 1933) Deprecated in Mac OS X v10.0
Performs background processing. (Deprecated. There is no replacement because this function does nothing in Mac OS X.)

`ATSUSetTextHandleLocation` (page 1963) Deprecated in Mac OS X v10.0
Associates text with a text layout object. (Deprecated. Use `ATSUSetTextPointerLocation` (page 1965) instead. See the Discussion for more details.)

Functions

ATSUBatchBreakLines

Calculates soft line breaks for the text associated with a text layout object.

```
OSStatus ATSUBatchBreakLines (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iRangeStart,
   UniCharCount iRangeLength,
   ATSUTextMeasurement iLineWidth,
   ItemCount *oBreakCount
);
```

Parameters

iTextLayout

The `ATSUTextLayout` for which you want to determine soft line breaks.

iRangeStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the text range to examine. To specify the beginning of the text buffer, pass the constant `kATSUFromTextBeginning`.

iRangeLength

The number of characters in which to consider in the determination of the soft line breaks.

iLineWidth

An `ATSUTextMeasurement` value specifying the line width for the text, as measured from the offset provided in the `iLineStart` parameter. You must pass a nonzero value. You should use the same width as the width layout control set for the text layout object since the final layout of each line is based on the controls set for the line or the entire text layout object. If no line width has been set for the line, `ATSUBatchBreakLines` uses the line width set for the text layout object; if this value is not set, `ATSUBatchBreakLines` returns `paramErr`.

Note that the value you pass for the `iLineWidth` parameter is used only for the line-breaking operation. For justification, flushness, and other operations to work properly you must also use this value as the line width for the text layout object. You can set the line width for the text layout object by calling the function `ATSUSetLineControls` or `ATSUSetLayoutControls` with the `kATSULineWidthTag` and the line width value.

oBreakCount

The number of soft line breaks found and set from the call. If you do not want to obtain the number of soft line breaks, then set this parameter to `NULL`.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUBatchBreakLines` function is equivalent to repeatedly calling the `ATSUBreakLine` function with the parameter `iUseAsSoftLineBreak` set to `true`. However the `ATSUBatchBreakLines` function performs more efficiently than repeated call to the `ATSUBreakLine` function.

You must call the `ATSUGetSoftLineBreaks` function to obtain the actual soft line breaks that were determined and set by the `ATSUBatchBreakLines` function.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeDrawing.h`

ATSUBreakLine

Calculates and, optionally, sets a soft line break in a range of text.

```
OSStatus ATSUBreakLine (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart,
   ATSUTextMeasurement iLineWidth,
   Boolean iUseAsSoftLineBreak,
   UniCharArrayOffset *oLineBreak
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

iLineStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the text range to examine. To specify the beginning of the text buffer, pass the constant `kATSUFromTextBeginning`. When calling `ATSUBreakLine` repeatedly to obtain all the soft line breaks for a given text range, in each subsequent call pass the value produced in the `oLineBreak` parameter by the immediately prior call to `ATSUBreakLine`.

iLineWidth

An `ATSUTextMeasurement` value specifying the line width for the text, as measured from the offset provided in the `iLineStart` parameter. You must pass a nonzero value. You can pass `kATSUUseLineControlWidth` to indicate that `ATSUBreakLine` should use the previously set line width attribute for the current line to determine how many characters can fit on the line. If no line width has been set for the line, `ATSUBreakLine` uses the line width set for the text layout object; if this value is not set, `ATSUBreakLine` returns `paramErr`.

Note that the value you pass for the `iLineWidth` parameter is used only for the line-breaking operation. For justification, flushness, and other operations to work properly you must also use this value as the line width for the text layout object. You can set the line width for the text layout object by calling the function `ATSUSetLineControls` or `ATSUSetLayoutControls` with the `kATSULineWidthTag` and the line width value.

iUseAsSoftLineBreak

A `Boolean` value indicating whether `ATSUBreakLine` should automatically set the line break produced in the `oLineBreak` parameter. If `true`, `ATSUBreakLine` sets the line break and clears any previously-set soft line breaks that precede the new break in the line but lie after the offset specified by `iLineStart`.

oLineBreak

A pointer to a `UniCharArrayOffset` value. On return, the value specifies the offset from the beginning of the text layout object's text buffer to the location of the calculated soft line break. If the value produced is the same value as specified in `iLineStart`, you have made an input parameter error. In this case, check to make sure that the line width specified in `iLineWidth` is big enough for `ATSUBreakLine` to perform line breaking. `ATSUBreakLine` does not return an error in this case. ATSUI usually calculates a soft line break to be at the beginning of the first word that does not fit on the line. But if `ATSUBreakLine` calculates the most optimal line break to be in the middle of a word, it returns the result code `kATSULineBreakInWord`. Note that ATSUI produces a line break in the middle of a word only as a last resort.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion
When the user inserts or deletes text or changes text layout attributes that affect how glyphs are laid out, you must determine whether the affected range of text still fits in the set line width, that is, whether the text needs to be rewrapped. You can use the `ATSUBreakLine` function to calculate a soft line break, based on the line width and text range you specify. If you pass `true` for `iUseAsSoftLineBreak`, `ATSUBreakLine` sets the soft line break it calculates and performs line layout on the characters.

If you need to calculate and set soft line breaks for a range of text and you want to use the same width for all lines in this range, you should call the function `ATSUBatchBreakLines` (page 1844). Calling `ATSUBatchBreakLines` is equivalent to repeatedly calling the `ATSUBreakLine` function with the parameter `iUseAsSoftLineBreak` set to true. However, the `ATSUBatchBreakLines` function performs more efficiently than repeated calls to the `ATSUBreakLine` function.

If you do choose to call the `ATSUBreakLine` function repeatedly to obtain all possible line breaks for a range of text it will produce the previously set soft line break(s) if there are no additional line breaks to be found, or if the user has altered the text range or its attributes in a way that does not affect glyph layout.

The `ATSUBreakLine` function suggests a soft line break each time it encounters a hard line break character such as a carriage return, line feed, form feed, line separator, or paragraph separator. If `ATSUBreakLine` does not encounter a hard line break, it uses the line width you specify to determine how many characters fit on a line and suggests soft line breaks accordingly.

If you pass `true` for `iUseAsSoftLineBreak`, `ATSUBreakLine` uses the soft line break it calculates to perform line layout on the characters. `ATSUBreakLine` then determines whether the characters still fit within the line, which is necessary due to end-of-line effects such as swashes. When `ATSUBreakLine` sets a soft line break, it clears any previously-set soft line breaks that precede the new break in the line but lie after the offset specified by `iLineStart`.

Before calculating soft line breaks, `ATSUBreakLine` turns off any previously set line justification, rotation, width, alignment, descent, and ascent values and treats the text as a single line. Additionally, `ATSUBreakLine` examines the text layout object to ensure that each of the characters in the range is assigned to a style run. If there are gaps between style runs, `ATSUBreakLine` assigns the characters in the gap to the style run that precedes (in storage order) the gap. If there is no style run at the beginning of the text range, `ATSUBreakLine` assigns these characters to the first style run it finds. If there no style run at the end of the text range, `ATSUBreakLine` assigns the remaining characters to the last style run it finds.

For optimal performance, you should use `ATSUBreakLine` or `ATSUBatchBreakLines` to both calculate and set soft line breaks in your text. You should typically only call the function `ATSUSetSoftLineBreak` (page 1961) to set soft line breaks when you are using your own line-breaking algorithm to calculate soft line breaks.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUCalculateBaselineDeltas

Obtains the optimal baseline positions for glyphs in a style run.

```
OSStatus ATSUCalculateBaselineDeltas (
   ATSUStyle iStyle,
   BslnBaselineClass iBaselineClass,
   BslnBaselineRecord oBaselineDeltas
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object to examine.

iBaselineClass

A `BslnBaselineClass` constant identifying the primary baseline from which to measure other baselines. See `SFNTLayoutTypes.h` for an enumeration of possible values. Pass the constant `kBSLNNoBaselineOverride` to use the standard baseline value from the current font.

oBaselineDeltas

A `BslnBaselineRecord` array consisting of `Fixed` values. On return, the array contains baseline offsets, specifying distances measured in points, from the default baseline to each of the other baseline types in the style object. Positive values indicate baselines above the default baseline and negative values indicate baselines below it. See `SFNTLayoutTypes.h` for a description of the `BslnBaselineRecord` type.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

Depending on the writing system, a baseline may be above, below, or through the centers of glyphs. In general, a style run has a default baseline, to which all glyphs are visually aligned when the text is laid out. For example, in a run of Roman text, the default baseline is the Roman baseline, upon which glyphs sit (except for descenders, which extend below the baseline).

You can call the `ATSUCalculateBaselineDeltas` function to obtain the distances from a specified baseline type to that of other baseline types for a given style object. `ATSUCalculateBaselineDeltas` takes into account font and text size when performing these calculations. ATSUI uses these distances to determine the cross-stream shifting to apply when aligning glyphs in a style run. You can use the resulting array to set or obtain the optimal baseline positions of glyphs in a style run. You can also set various baseline values to create special effects such as drop capitals.

The functions `ATSUSetLineControls` (page 1956) and `ATSUSetLayoutControls` (page 1955) allow you to set baseline offset values at the line or layout level, respectively, using the `kATSULineBaselineValuesTag` control attribute tag. For more information on `kATSULineBaselineValuesTag`, see "Attribute Tags" (page 2030).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUClearAttributes

Restores default values to the specified style attributes of a style object.

```
OSStatus ATSUClearAttributes (
   ATSUStyle iStyle,
   ItemCount iTagCount,
   const ATSUAttributeTag iTag[]
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to restore default style attribute values.

iTagCount

An `ItemCount` value specifying the number of attributes to restore to default values. This value should correspond to the number of elements in the `iTag` array. To restore all style attributes in the specified style object, pass the constant `kATSUClearAll` in this parameter. In this case, the value in the `iTag` parameter is ignored.

iTag

A pointer to the initial `ATSUAttributeTag` constant in an array of attribute tags. Each tag should identify a style attribute to restore to its default value. See "Attribute Tags" (page 2030) for a description of the Apple-defined style attribute tag constants.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUClearAttributes` function removes those style attribute values identified by the tag constants in the `iTag` array and replaces them with the default values described in "Attribute Tags" (page 2030). If you specify that any currently unset attribute values be removed, the function does not return an error.

To remove all previously set style attribute, font feature, and font variation values from a style object, call the function `ATSUClearStyle` (page 1854).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUClearFontFeatures

Restores default settings to the specified font features of a style object.

```
OSStatus ATSUClearFontFeatures (
   ATSUStyle iStyle,
   ItemCount iFeatureCount,
   const ATSUFontFeatureType iType[],
   const ATSUFontFeatureSelector iSelector[]
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to restore default font feature settings.

iFeatureCount

An `ItemCount` value specifying the number of font features to restore to default settings. This value should correspond to the number of elements in the *iType* and *iSelector* arrays. To restore default settings to all the font features in the specified style object, pass the constant `kATSUClearAll` in this parameter. In this case, the values in the `iType` and `iSelector` parameters are ignored.

iType

A pointer to the initial `ATSUFontFeatureType` value in an array of feature types. Each value should identify a font feature to restore to its default setting. To obtain all previously set font features for a given style object, you can call the function `ATSUGetAllFontFeatures` (page 1887).

iSelector

A pointer to the initial `ATSUFontFeatureSelector` value in an array of feature selectors. Each element in the array must contain a valid feature selector corresponding to a font feature you provide in the *iType* parameter. To obtain all previously set feature selectors for a given style object, you can call the function `ATSUGetAllFontFeatures` (page 1887).

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUClearFontFeatures` function removes those font features that are identified by the feature selector and type constants in the `iSelector` and `iType` arrays and replaces them with their font-defined default values. Note that if you pass `ATSUClearFontFeatures` a font feature and selector that are already at default settings, the function does not return an error.

To restore default font variations to a style object, call the function `ATSUClearFontVariations` (page 1850). To restore default style attributes to a style object, call `ATSUClearAttributes` (page 1848). To restore all default settings to a style object (for font features, variations, and style attributes), call the function `ATSUClearStyle` (page 1854).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUClearFontVariations

Restores default values to the specified font variation axes of a style object.

```
OSStatus ATSUClearFontVariations (
   ATSUStyle iStyle,
   ItemCount iAxisCount,
   const ATSUFontVariationAxis iAxis[]
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to restore default font variation axis settings.

iAxisCount

An ItemCount value specifying the number of font variation axes to restore to default settings. This value should correspond to the number of elements in the *iAxis* array. To restore default values to all the font variation axes in the style object, pass the constant kATSUClearAll in this parameter. If you pass kATSUClearAll the value in the iAxis parameter is ignored.

iAxis

A pointer to the initial ATSUFontVariationAxis tag in an array of font variation axes. Each element in the array must contain a valid tag that corresponds to a font variation axis to restore to its default setting. You can obtain variation axis tags for a style object from the function ATSUGetAllFontVariations (page 1888).

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The ATSUClearFontVariations function removes those font variation axis values identified by variation axis tags in the iAxis array and replaces them with their font-defined default values. You can remove unset font variation values from a style object without a function error.

To restore default font features to a style object, call the function ATSUClearFontFeatures (page 1849). To restore default style attributes, call ATSUClearAttributes (page 1848). To restore all default settings to a style object (for font features, variations, and style attributes), call the function ATSUClearStyle (page 1854).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeFonts.h

ATSUClearLayoutCache

Clears the layout cache of a line or an entire text layout object.

```
OSStatus ATSUClearLayoutCache (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart
);
```

Parameters

iTextLayout

An ATSUTextLayout value identifying the text layout object for which to clear a layout cache.

iLineStart

A UniCharArrayOffset value specifying the offset from the beginning of the text buffer to the beginning of the line for which to discard the layout cache. If the range of text spans multiple lines, you should call ATSUClearLayoutCache for each line, passing the offset corresponding to the beginning of the new line to draw with each call. To clear the layout cache of the entire text layout object, you can pass the constant kATSUFromTextBeginning.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The layout cache contains all the layout information ATSUI calculates and needs to draw a range of text in a text layout object. This includes caret positions, the memory locations of glyphs, and other information needed to lay out the glyphs. ATSUI uses information in the layout cache to avoid laying out the text again, thereby improving performance. When you clear the layout cache of a line or block of text, ATSUI takes longer to redraw a line, since it must perform the calculations that support glyph layout again.

You should call the function `ATSUClearLayoutCache` when you need to decrease the amount of memory your application uses. This function reclaims memory at the cost of optimal performance.

By default, the `ATSUClearLayoutCache` function removes the layout cache of a single line. To clear the layout cache for multiple lines, you should call `ATSUClearLayoutCache` for each line. To clear the layout cache of an entire text layout object, pass the constant `kATSUFromTextBeginning` in the `iLineStart` parameter. Note that `ATSUClearLayoutCache` does not produce a function error if lines do not have a layout cache.

The `ATSUClearLayoutCache` function flushes the layout cache but does not alter previously set text layout attributes, soft line break positions, or the text memory location. If you do not want to retain these values, you should dispose of the text layout object by calling the `ATSUDisposeTextLayout` (page 1876) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeObjects.h`

ATSUClearLayoutControls

Restores default values to the specified layout control attributes of a text layout object.

```
OSStatus ATSUClearLayoutControls (
   ATSUTextLayout iTextLayout,
   ItemCount iTagCount,
   const ATSUAttributeTag iTag[]
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object for which to restore default layout control attribute values.

iTagCount
An `ItemCount` value specifying the number of layout control attributes to restore to default values. This value should correspond to the number of elements in the `iTag` array. To restore all layout control attributes in the specified text layout object, pass the constant `kATSUClearAll` in this parameter. In this case, the value in the `iTag` parameter is ignored.

iTag
A pointer to the initial `ATSUAttributeTag` constant in an array of attribute tags. Each tag should identify a layout control attribute to restore to its default value. See "Attribute Tags" (page 2030) for a description of the Apple-defined layout control attribute tag constants.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUClearLayoutControls` function removes those layout control attribute values identified by the tag constants in the `iTag` array and replaces them with the default values described in "Attribute Tags" (page 2030). If you specify that any currently unset attribute values be removed, the function does not return an error.

To restore default values to line control attributes in a text layout object, call the function `ATSUClearLineControls` (page 1853).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUClearLineControls

Restores default values to the specified line control attributes of a line in a text layout object.

```
OSStatus ATSUClearLineControls (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart,
   ItemCount iTagCount,
   const ATSUAttributeTag iTag[]
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object containing the line for which to restore default line control attribute values.

iLineStart
A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the line for which to restore attribute values.

iTagCount
An `ItemCount` value specifying the number of line control attributes to restore to default values. This value should correspond to the number of elements in the `iTag` array. To restore all line control attributes of the specified line, pass the constant `kATSUClearAll` in this parameter. In this case, the value in the `iTag` parameter is ignored.

iTag
A pointer to the initial `ATSUAttributeTag` constant in an array of attribute tags. Each tag should identify a line control attribute to restore to its default value. See "Attribute Tags" (page 2030) for a description of the Apple-defined line control attribute tag constants.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUClearLineControls` function removes those line control attribute values identified by the tag constants in the `iTag` array and replaces them with the default values described in "Attribute Tags" (page 2030). If you specify that any currently unset attribute values be removed, the function does not return an error.

To restore default values to layout control attributes in a text layout object, call the function `ATSUClearLayoutControls` (page 1852).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUClearSoftLineBreaks

Removes soft line breaks from a range of text.

```
OSStatus ATSUClearSoftLineBreaks (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iRangeStart,
   UniCharCount iRangeLength
);
```

Parameters

`iTextLayout`
An `ATSUTextLayout` value specifying the text layout object for which to remove line breaks.

`iRangeStart`
A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the text range. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iRangeLength` parameter.

`iRangeLength`
A `UniCharCount` value specifying the length of the text range. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUClearSoftLineBreaks` function clears all previously set soft line breaks for the specified text range and clears any associated layout caches as well.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUClearStyle

Restores default values to a style object.

```
OSStatus ATSUClearStyle (
   ATSUStyle iStyle
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to restore default values.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUClearStyle` function clears a style object of all style attributes (including any application-defined attributes), font features, and font variations and returns these values to their default settings. Default font variations and font features are defined by the font; default style attribute values are described in "Attribute Tags" (page 2030). `ATSUClearStyle` does not remove reference constants.

To restore only default style attributes to a style object, you should call the function `ATSUClearAttributes` (page 1848). To restore only font variations to a style object, call `ATSUClearFontVariations` (page 1850). To restore only font features, call `ATSUClearFontFeatures` (page 1849).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUCompareStyles

Compares the attribute values of two style objects.

```
OSStatus ATSUCompareStyles (
   ATSUStyle iFirstStyle,
   ATSUStyle iSecondStyle,
   ATSUStyleComparison *oComparison
);
```

Parameters

iFirstStyle

An `ATSUStyle` value specifying the first style object to compare.

iSecondStyle

An `ATSUStyle` value specifying the second style object to compare.

oComparison

A pointer to an `ATSUStyleComparison` value. On return, the value contains the results of the comparison and indicates whether the two style objects are the same, different, or one a subset of the another. See "Style Comparison Options" (page 2063) for a description of possible values.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCompareStyles` function compares the contents of two style objects, including their style attributes, font features, and font variations. It does not consider reference constants or application-defined style attributes in the comparison.

You can call `ATSUCompareStyles`, in conjunction with the function `ATSUGetAllAttributes` (page 1886), to implement style sheets and tables of style runs.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCopyAttributes

Copies all style attribute settings from a source style object to a destination style object.

```
OSStatus ATSUCopyAttributes (
   ATSUStyle iSourceStyle,
   ATSUStyle iDestinationStyle
);
```

Parameters

`iSourceStyle`
An `ATSUStyle` value specifying the style object from which to copy style attributes.

`iDestinationStyle`
An `ATSUStyle` value specifying the style object to set style attributes to.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCopyAttributes` function copies all style attributes to a destination style object from a source style object, including any default values (those values not set by your application) in the source object. Default values for style attributes are described in "Attribute Tags" (page 2030).

The `ATSUCopyAttributes` function does not copy the contents of memory referenced by pointers within custom style attributes or within reference constants. You are responsible for ensuring that this memory remains valid until both the source and destination style objects are disposed of.

To copy style attributes that are explicitly set in the source but not in the destination style object, call the function `ATSUUnderwriteAttributes` (page 1972). To copy all style attributes that are explicitly set in the source object into the destination object, whether or not the destination object has its own settings for these values, call the function `ATSUOverwriteAttributes` (page 1944).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCopyLayoutControls

Copies all layout control attribute settings from a source text layout object to a destination text layout object.

```
OSStatus ATSUCopyLayoutControls (
   ATSUTextLayout iSourceTextLayout,
   ATSUTextLayout iDestTextLayout
);
```

Parameters

iSourceTextLayout

An `ATSUTextLayout` value specifying the text layout object from which to copy layout control attributes.

iDestTextLayout

An `ATSUTextLayout` value specifying the text layout object for which to set layout control attributes.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUCopyLayoutControls` function copies all layout control attribute values to a destination text layout object from a source text layout object, including any default (unset) values in the source object. Default values for unset layout control attributes are described in "Attribute Tags" (page 2030).

`ATSUCopyLayoutControls` does not copy the contents of memory referenced by pointers within reference constants. You are responsible for ensuring that this memory remains valid until both the source and destination text layout objects are disposed.

To copy line control attribute values from one text layout object to another, call the function `ATSUCopyLineControls` (page 1857).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUCopyLineControls

Copies line control attribute settings from a line in a source text layout object to a line in a destination text layout object.

```
OSStatus ATSUCopyLineControls (
   ATSUTextLayout iSourceTextLayout,
   UniCharArrayOffset iSourceLineStart,
   ATSUTextLayout iDestTextLayout,
   UniCharArrayOffset iDestLineStart
);
```

Parameters

iSourceTextLayout

An `ATSUTextLayout` value specifying the text layout object from which to copy line control attributes.

iSourceLineStart
A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the line from which to copy control attributes.

iDestTextLayout
An `ATSUTextLayout` value specifying the text layout object for which to set line control attributes. This can be the same text layout object passed in the `iSourceTextLayout` parameter if you want to copy line control attributes from one line to another within a text layout object.

iDestLineStart
A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the line for which to set control attributes.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCopyLineControls` function copies all line control attribute values to a line in a destination text layout object from a line in a source text layout object, including any default (unset) values in the source line. Unset line control attributes are assigned the default values described in "Attribute Tags" (page 2030).

`ATSUCopyLineControls` does not copy the contents of memory referenced by pointers within reference constants. You are responsible for ensuring that this memory remains valid until the source text layout object is disposed.

To copy layout control attributes from one text layout object to another, call the function `ATSUCopyLayoutControls` (page 1857).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCopyToHandle

Copies an ATSUI style to a handle. (Deprecated in Mac OS X v10.1. Use `ATSUFlattenStyleRunsToStream` (page 1882) instead.)

Not Recommended

```
OSStatus ATSUCopyToHandle (
   ATSUStyle iStyle,
   Handle oStyleHandle
);
```

Parameters
iStyle
An `ATSUStyle` value.

oStyleHandle
A valid handle.

Return Value
A result code.

Discussion
The `ATSUCopyToHandle` function is not recommended for use, as this function does not produce the correct data format for the display of ATSUI style data. You should instead use the function `ATSUFlattenStyleRunsToStream` to flatten style data and the function `ATSUUnflattenStyleRunsFromStream` to unflatten style data. These functions read and write data using the `ustl` data specification. You can use a data block format of this type to copy and paste Unicode-encoded styled text between applications or within your application. The `ustl` data structure contains flattened text layout data, flattened style run data, and flattened style list data. For more information on the `ustl` data structure see *Inside Mac OS X: ATSUI Reference*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.1.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFlattening.h`

ATSUCountFontFeatureSelectors

Obtains the number of available feature selectors for a given feature type in a font.

```
OSStatus ATSUCountFontFeatureSelectors (
   ATSUFontID iFontID,
   ATSUFontFeatureType iType,
   ItemCount *oSelectorCount
);
```

Parameters

iFont
An `ATSUFontID` value identifying the font to examine.

iType
An `ATSUFontFeatureType` value specifying one of the font's supported feature types. To obtain the available feature types for a font, call the function `ATSUGetFontFeatureTypes` (page 1898).

oSelectorCount
A pointer to an `ItemCount` value. On return, the value specifies the actual number of feature selectors defined for the feature type by the font.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCountFontFeatureSelectors` function obtains the total number of feature selectors defined for a given feature type in the font. You can use the count produced by `ATSUCountFontFeatureSelectors` to determine how much memory to allocate for the `oSelectors` array in the function `ATSUGetFontFeatureSelectors` (page 1897).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUCountFontFeatureTypes

Obtains the number of available feature types in a font.

```
OSStatus ATSUCountFontFeatureTypes (
   ATSUFontID iFontID,
   ItemCount *oTypeCount
);
```

Parameters

iFont

An `ATSUFontID` value identifying the font to examine.

oTypeCount

A pointer to an `ItemCount` value. On return, the value specifies the actual number of feature types defined for the font.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUCountFontFeatureTypes` function obtains the total number of feature types defined for a font. You can use the count produced by `ATSUCountFontFeatureTypes` to determine how much memory to allocate for the `oTypes` array in the function `ATSUGetFontFeatureTypes` (page 1898).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeFonts.h`

ATSUCountFontInstances

Obtains the number of defined font instances in a font.

```
OSStatus ATSUCountFontInstances (
   ATSUFontID iFontID,
   ItemCount *oInstances
);
```

Parameters

iFont

An `ATSUFontID` value identifying the font to examine.

oInstances

A pointer to an `ItemCount` value. On return, the value specifies the number of font instances defined for the font.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUCountFontInstances` function obtains the total number of font instances defined in a font. You can use an index value derived from this count to get information about a specific font instance by calling the function `ATSUGetFontInstance` (page 1900).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUCountFontNames

Obtains the number of font names that correspond to a given ATSUI font ID.

```
OSStatus ATSUCountFontNames (
   ATSUFontID iFontID,
   ItemCount *oFontNameCount
);
```

Parameters

iFontID

An `ATSUFontID` value specifying the font to examine.

oFontNameCount

A pointer to an `ItemCount` value. On return, the value specifies the number of entries in the font name table corresponding to the given ATSUI font ID.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCountFontNames` function obtains the number of font names defined in a font name table for a given ATSUI font ID. This number includes repetitions of the same name in different platforms, languages, and scripts; names of font features, variations, tracking settings, and instances for the font; and font names identified by name code constants.

You can pass an index value based on this count to the function `ATSUGetIndFontName` (page 1908) to obtain a name string, name code, platform, script, and language for a given ATSUI font ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUCountFontTracking

Obtains the number of entries in the font tracking table that correspond to a given ATSUI font ID and glyph orientation.

```
OSStatus ATSUCountFontTracking (
   ATSUFontID iFontID,
   ATSUVerticalCharacterType iCharacterOrientation,
   ItemCount *oTrackingCount
);
```

Parameters

iFont

An `ATSUFontID` value specifying the font to examine.

iCharacterOrientation

An `ATSUVerticalCharacterType` constant identifying the glyph orientation of the font tracking entries, for example `kATSUStronglyHorizontal` or `kATSUStronglyVertical`. See "Vertical Character Types" (page 2068) for a description of possible values.

oTrackingCount

A pointer to an `ItemCount` value. On return, the value specifies the number of entries in the font tracking table corresponding to the given ATSUI font ID and glyph orientation.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUCountFontTracking` function obtains the number of font tracking entries defined in a font tracking table for a given ATSUI font ID and glyph orientation. You can pass an index value based on this count to the function `ATSUGetIndFontTracking` (page 1910) to obtain the name code and tracking value of a font tracking.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeFonts.h`

ATSUCountFontVariations

Obtains the number of defined variation axes in a font.

```
OSStatus ATSUCountFontVariations (
   ATSUFontID iFontID,
   ItemCount *oVariationCount
);
```

Parameters

iFont

An `ATSUFontID` value identifying the font to examine.

oVariationCount

A pointer to an `ItemCount` value. On return, the value specifies the number of variation axes defined for the font.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCountFontVariations` function obtains the total number of variation axes defined for a font. You can use the count produced by `ATSUCountFontVariations` to get information about a specific font variation axis from the function `ATSUGetIndFontVariation` (page 1911).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUCreateAndCopyStyle

Creates a copy of a style object.

```
OSStatus ATSUCreateAndCopyStyle (
   ATSUStyle iStyle,
   ATSUStyle *oStyle
);
```

Parameters

`iStyle`
An `ATSUStyle` value specifying the style object to copy.

`oStyle`
A pointer to an `ATSUStyle` value. On return, the pointer refers to a newly created style object. This style object contains the same values for style attributes, font features, and font variations as those of the style object passed in the `iStyle` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCreateAndCopyStyle` function creates a new style object with values obtained from the source style object's style attributes, font features, and font variations. `ATSUCreateAndCopyStyle` does not copy reference constants.

To create a new style object without copying a source object, you can call the function `ATSUCreateStyle` (page 1865). Alternately, to copy the contents of a source style object into an existing style object, call the function `ATSUCopyAttributes` (page 1856).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCreateAndCopyTextLayout

Creates a copy of a text layout object.

```
OSStatus ATSUCreateAndCopyTextLayout (
   ATSUTextLayout iTextLayout,
   ATSUTextLayout *oTextLayout
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to copy.

oTextLayout

A pointer to an `ATSUTextLayout` value. On return, the pointer refers to a newly created text layout object containing the contents of the text layout object in the `iTextLayout` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCreateAndCopyTextLayout` function creates a copy of the source text layout object's style runs (including references to the associated text buffer and style objects), line attributes, layout attributes, and layout caches. `ATSUCreateAndCopyTextLayout` does not copy reference constants.

To create a text layout object without copying a source object, you can the function `ATSUCreateTextLayout` (page 1866) or the function `ATSUCreateTextLayoutWithTextPtr` (page 1869).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCreateFontFallbacks

Creates an opaque object that can be set to contain a font list and a font-search method.

```
OSStatus ATSUCreateFontFallbacks (
   ATSUFontFallbacks *oFontFallback
);
```

Parameters

oFontFallback

A pointer to an `ATSUFontFallbacks` value. On return, the pointer refers to a newly created font fallback object.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCreateFontFallbacks` function creates an "empty" font fallback object, which can be used to define ATSUI's search behavior when seeking substitute fonts for a text layout object. Font fallback objects are thread safe and can be shared among threads.

You set the font list and search method for the font fallback object by calling the function `ATSUSetObjFontFallbacks` (page 1958). To associate the font fallback object with a text layout object, call either of the functions `ATSUSetLayoutControls` (page 1955) or `ATSUSetLineControls` (page 1956). You pass these functions the control attribute value `kATSULineFontFallbacksTag` to set the font fallback object.

Similarly to a style object, a font fallback object can be used with any number of text layout objects. While it is innately more efficient to reuse font fallback objects, instead of repeatedly creating (and destroying) them, there is another reason to share a given font fallback object among text layout objects. That is, as a font fallback object is used, it continues to amass data about the system's fonts and which are best applied to the various ranges of Unicode. Therefore, for best performance, once you create a font fallback object, you should keep it and use it as often as needed.

You should dispose of a font fallback object only when it is no longer needed in your application. To dispose of the memory associated with a font fallback object, call the function `ATSUDisposeFontFallbacks` (page 1875).

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCreateStyle

Creates an opaque style object containing only default style attributes, font features, and font variations.

```
OSStatus ATSUCreateStyle (
   ATSUStyle *oStyle
);
```

Parameters

oStyle
A pointer to an `ATSUStyle` value. On return, the pointer refers to an empty style object.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCreateStyle` function creates a style object containing only default values for style attributes, font features, and font variations. The default values for the font features and variations are assigned by the font. The default style attribute values are described in "Attribute Tags" (page 2030).

To make changes to the default style attribute values, you can call the function `ATSUSetAttributes` (page 1950). To set font features and font variations, call the functions `ATSUSetFontFeatures` (page 1952) and `ATSUSetVariations` (page 1967), respectively. You can also use the function `ATSUCreateAndCopyStyle` (page 1863) to create a new style object by copying all the settings from an existing one.

For ATSUI to apply your selected character-style information, you must associate the style object with a text run in a text layout object. A text run consists of one or more characters that are contiguous in memory. If you associate these characters with a distinct style, you define a style run. You can use the function `ATSUSetRunStyle` (page 1959) to define a style run by associating a style object with a run of text in a text

layout object. Or, to create a text layout object and associate style objects with it at the same time, you can call the function `ATSUCreateTextLayoutWithTextPtr` (page 1869). In either case, each text run in a text layout object must be assigned a style object, which may or may not differ from other style objects assigned to other text runs in the text layout object.

Style objects are readily reusable and should be cached for later use, if possible. You can create a style object once and then use it for as many text layout objects as appropriate. Style objects are thread-safe starting with ATSUI version 2.3.

Note that you are responsible for disposing of the memory allocated for the style object. However, you should dispose of any text layout objects with which the style object is associated prior to disposing of the style object itself. To dispose of a style object, call the function `ATSUDisposeStyle` (page 1876).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeObjects.h`

ATSUCreateTextLayout

Creates an opaque text layout object containing only default text layout attributes.

```
OSStatus ATSUCreateTextLayout (
   ATSUTextLayout *oTextLayout
);
```

Parameters

oTextLayout

A valid pointer to an `ATSUTextLayout` value. On return, the value refers to an empty text layout object.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCreateTextLayout` function creates a text layout object containing only the default text layout attributes described in "Attribute Tags" (page 2030). The resulting text layout object is associated with neither text nor style objects. However, most ATSUI functions that operate on text layout objects require that the objects be associated with style information and text. To associate style objects and text with an empty text layout object, you can call the functions `ATSUSetRunStyle` (page 1959) and `ATSUSetTextPointerLocation` (page 1965). Or, to create a text layout object and associate style objects and text with it at the same time, you can call the function `ATSUCreateTextLayoutWithTextPtr` (page 1869).

To provide nondefault line or layout attributes for a text layout object, you can call the functions `ATSUSetLineControls` (page 1956) or `ATSUSetLayoutControls` (page 1955). After setting text attributes, call `ATSUDrawText` (page 1877) to draw the text.

Text layout objects are readily reusable and should be cached for later use, if possible. You can reuse a text layout object even if the text associated with it is altered. Call the functions `ATSUSetTextPointerLocation` (page 1965), `ATSUTextDeleted` (page 1969), or `ATSUTextInserted` (page 1970) to manage the altered text.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCreateTextLayoutWithTextHandle

Creates an opaque text layout object containing default text layout attributes as well as associated text and text styles. (Deprecated in Mac OS X v10.0. Use `ATSUCreateTextLayoutWithTextPtr` (page 1869) instead. See the Discussion for more details.)

Not recommended.

```
OSStatus ATSUCreateTextLayoutWithTextHandle (
   UniCharArrayHandle iText,
   UniCharArrayOffset iTextOffset,
   UniCharCount iTextLength,
   UniCharCount iTextTotalLength,
   ItemCount iNumberOfRuns,
   const UniCharCount iRunLengths[],
   ATSUStyle iStyles[],
   ATSUTextLayout *oTextLayout
);
```

Parameters

iText

A handle of type `UniCharArrayHandle` referring to a text buffer containing UTF-16–encoded text. ATSUI associates this buffer with the new text layout object and analyzes the entire text of the buffer when obtaining the layout context for the current text range. Thus, for paragraph-format text, if you specify a buffer containing less than a complete paragraph, some of ATSUI's layout results are not guaranteed to be accurate. For example, with a buffer of less than a full paragraph, ATSUI can neither reliably obtain the context for bidirectional processing nor reliably generate accent attachments and ligature formations for Roman text.

iTextOffset

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the range to include in the layout. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iTextLength` parameter.

iTextLength

A `UniCharCount` value specifying the length of the text range. Note that `iTextOffset + iTextLength` must be less than or equal to the value of the *iTextTotalLength* parameter. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

iTextTotalLength

A `UniCharCount` value specifying the length of the entire text buffer. This value should be greater than or equal to the range of text defined by the `iTextLength` parameter.

iNumberOfRuns

An `ItemCount` value specifying the number of text style runs you want to define within the text range. The number of style objects and style run lengths passed in the `iStyles` and `iRunLengths` parameters, respectively, should each be equal to the number of runs specified here.

iRunLengths

A pointer to a `UniCharCount` array specifying the lengths of each style run in the text layout object. You can pass `kATSUToTextEnd` for the last style run length if you want the style run to extend to the end of the text range. If the sum of the style run lengths is less than the total length of the text range, the remaining characters are assigned to the last style run.

iStyles

A pointer to the first element in an `ATSUStyle` array. Each element in the array must contain a valid style object that corresponds to a style run defined by the `iRunLengths` array.

oTextLayout

A valid pointer to an `ATSUTextLayout` value. On return, the value refers to the newly created text layout object.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should use the function `ATSUCreateTextLayoutWithTextPtr` (page 1869) instead of using the function `ATSUCreateTextLayoutWithTextHandle`.

The `ATSUCreateTextLayoutWithTextHandle` function creates a text layout object associated with style objects and text and containing the default text layout attributes described in "Attribute Tags" (page 2030). To provide nondefault line or layout attributes for a text layout object, you can call the functions `ATSUSetLineControls` (page 1956) or `ATSUSetLayoutControls` (page 1955). After setting text attributes, call `ATSUDrawText` (page 1877) to draw the text.

Because the only way that ATSUI interacts with text is via the memory references you associate with a text layout object, you are responsible for keeping these references updated, as in the following cases:

1. When the user deletes or inserts a subrange within a text buffer (but the buffer itself is not relocated), you should call the functions `ATSUTextDeleted` (page 1969) and `ATSUTextInserted` (page 1970), respectively.

2. When you relocate the entire text buffer (but no other changes have occurred that would affect the buffer's current subrange), you should call the function `ATSUTextMoved` (page 1971).

3. When both the buffer itself is relocated and a subrange of the buffer's text is deleted or inserted (that is, a combination of cases 1 and 2, above), you must use either the function `ATSUSetTextHandleLocation` (page 1963) or the function `ATSUSetTextPointerLocation` (page 1965) to inform ATSUI.

4. When you are associating an entirely different buffer with a text layout object, you must call either the function `ATSUSetTextHandleLocation` (page 1963) or the function `ATSUSetTextPointerLocation` (page 1965).

Note that, because ATSUI objects retain state information, doing superfluous calling can degrade performance. For example, you could call `ATSUSetTextHandleLocation` rather than `ATSUTextInserted` when the user inserts text, but there would be a performance penalty, as all the layout caches are flushed when you call `ATSUSetTextHandleLocation`, rather than just the affected ones.

Text layout objects are readily reusable and should themselves be cached for later use, if possible.

The `ATSUCreateTextLayoutWithTextHandle` function associates text with a text layout object via a handle, but ATSUI functions that need to access the text return the handle to its original state upon completion.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.0.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUCreateTextLayoutWithTextPtr

Creates an opaque text layout object containing default text layout attributes as well as associated text and text styles.

```
OSStatus ATSUCreateTextLayoutWithTextPtr (
   ConstUniCharArrayPtr iText,
   UniCharArrayOffset iTextOffset,
   UniCharCount iTextLength,
   UniCharCount iTextTotalLength,
   ItemCount iNumberOfRuns,
   const UniCharCount iRunLengths[],
   ATSUStyle iStyles[],
   ATSUTextLayout *oTextLayout
);
```

Parameters

iText

A pointer of type `ConstUniCharArrayPtr`, referring to a text buffer containing UTF-16–encoded text. ATSUI associates this buffer with the new text layout object and analyzes the complete text of the buffer when obtaining the layout context for the current text range. Thus, for paragraph-format text, if you specify a buffer containing less than a complete paragraph, some of ATSUI's layout results are not guaranteed to be accurate. For example, with a buffer of less than a full paragraph, ATSUI can neither reliably obtain the context for bidirectional processing nor reliably generate accent attachments and ligature formations for Roman text.

iTextOffset

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the range to include in the layout. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iTextLength` parameter.

iTextLength

A `UniCharCount` value specifying the length of the text range. Note that `iTextOffset + iTextLength` must be less than or equal to the value of the *`iTextTotalLength`* parameter. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

iTextTotalLength

A `UniCharCount` value specifying the length of the entire text buffer. This value should be greater than or equal to the range of text defined by the `iTextLength` parameter.

iNumberOfRuns

An `ItemCount` value specifying the number of text style runs you want to define within the overall text range. The number of style objects and style run lengths passed in the `iStyles` and `iRunLengths` parameters, respectively, should be equal to the number of runs specified here.

iRunLengths

A pointer to the first element in a `UniCharCount` array. This array provides ATSUI with the lengths of each of the text's style runs. You can pass `kATSUToTextEnd` for the last style run length if you want the style run to extend to the end of the text range. If the sum of the style run lengths is less than the total length of the text range, the remaining characters are assigned to the last style run.

iStyles

A pointer to the first element in an `ATSUStyle` array. Each element in the array must contain a valid style object that corresponds to a style run defined by the `iRunLengths` array.

oTextLayout

A valid pointer to an `ATSUTextLayout` value. On return, the value refers to the newly created text layout object.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUCreateTextLayoutWithTextPtr` function creates a text layout object associated with style objects and text and containing the default text layout attributes described in "Attribute Tags" (page 2030). To provide nondefault line or layout attributes for a text layout object, you can call the functions `ATSUSetLineControls` (page 1956) or `ATSUSetLayoutControls` (page 1955). After setting text attributes, call `ATSUDrawText` (page 1877) to draw the text.

Because the only way that ATSUI interacts with text is via the memory references you associate with a text layout object, you are responsible for keeping these references updated, as in the following cases:

1. When the user deletes or inserts a subrange within a text buffer (but the buffer itself is not relocated), you should call the functions `ATSUTextDeleted` (page 1969) and `ATSUTextInserted` (page 1970), respectively.

2. When you relocate the entire text buffer (but no other changes have occurred that would affect the buffer's current subrange), you should call the function `ATSUTextMoved` (page 1971).

3. When both the buffer itself is relocated and a subrange of the buffer's text is deleted or inserted (that is, a combination of cases 1 and 2, above), you must use the function `ATSUSetTextPointerLocation` (page 1965) to inform ATSUI.

4. When you are associating an entirely different buffer with a text layout object, you must call the function `ATSUSetTextPointerLocation` (page 1965).

Note that, because ATSUI objects retain state information, doing superfluous calling can degrade performance. For example, you could call `ATSUSetTextPointerLocation` rather than `ATSUTextInserted` when the user inserts text, but there would be a performance penalty, as all the layout caches are flushed when you call `ATSUSetTextPointerLocation`, rather than just the affected ones.

Text layout objects are readily reusable and should themselves be cached for later use, if possible. Text objects are thread-safe starting with ATSUI version 2.4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeObjects.h

ATSUDirectAddStyleSettingRef

Looks up, and if necessary, adds a style setting to a line of text.

```
OSStatus ATSUDirectAddStyleSettingRef (
   ATSULineRef iLineRef,
   ATSUStyleSettingRef iStyleSettingRef,
   UInt16 *oStyleIndex
);
```

Parameters

iLineRef

An ATSULineRef value that specifies the line of text to which you want to add a style setting. You should pass the same reference provided as a parameter to your ATSUDirectLayoutOperationOverrideProcPtr (page 1998) callback function.

iStyleSettingRef

An ATSUStyleSettingRef value that specifies the style setting you want ATSUI to look up or add to the text layout object referenced by the line starting at the offset iLineOffset.

oStyleIndex

On return, points to the index of the ATSUStyleSettingRef passed in iStyleSettingRef for the line referenced by iLineRef. If the ATSUStyleSettingRef does not exist in that context, ATSUI adds it and returns the index value.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The function ATSUDirectAddStyleSettingRef checks to see if a line of text has a specified style setting reference associated with it. If the style setting reference is not associated with the line of text, ATSUI adds the style setting reference.

You must call this function from within an ATSUDirectLayoutOperationOverrideProcPtr (page 1998) callback function. You can use the function ATSUDirectAddStyleSettingRef to replace or substitute glyphs. For example, you can check a line of text for a specific character, such as a whitespace character. When your application finds a whitespace character, it can call the function ATSUDirectAddStyleSettingRef to set style attributes that achieve the desired effect.

Do not call this function if you obtained an ATSUStyleSettingRef array for the line specified by iLineRef and have not yet disposed of the pointer to this array by calling the function ATSUDirectReleaseLayoutDataArrayPtr (page 1874), as the pointer is not guaranteed to be valid after you call the function ATSUDirectAddStyleSettingRef.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeDirectAccess.h

ATSUDirectGetLayoutDataArrayPtrFromLineRef

Obtains the glyph data specified by a direct-data selector and for a specific line of text.

```
OSStatus ATSUDirectGetLayoutDataArrayPtrFromLineRef (
   ATSULineRef iLineRef,
   ATSUDirectDataSelector iDataSelector,
   Boolean iCreate,
   void *oLayoutDataArrayPtr[],
   ItemCount *oLayoutDataCount
);
```

Parameters

iLineRef

An `ATSULineRef` value that specifies the line of text whose data you want to obtain. You should pass the same `ATSULineRef` value passed to the `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998) callback function from which you are calling this function.

iDataSelector

A direct-data selector constant that specifies the data you want to obtain. You can pass any of the constants described in "Direct Data Selectors" (page 2044).

iCreate

A `Boolean` value that specifies whether to create an array if one does not already exist. Pass `true` if you want an array created. If the line referenced by the `iLineRef` parameter does not already have an array created that contains the data specified by the `iDataSelector` parameter, then ATSUI creates a zero-filled array and returns the array in the `oLayoutDataArray` parameter. The `iCreate` parameter has no effect for some data specified by the direct-data selector. See "Direct Data Selectors" (page 2044) for details.

oLayoutDataArrayPtr[]

On return, points to an array that contains the data specified by the `iDataSelector` parameter. The data is for the line of text referenced by the `iLineRef` parameter. If an array for the specified data does not exist, and if the `iCreate` is set to `false`, ATSUI returns `NULL`. If an array for the specified data does not exist, and if the `iCreate` is set to `true`, ATSUI creates a zero-filled array. You can pass `NULL` if you only want to obtain the number of entries in the array returned in the *oLayoutDataArray* array.

oLayoutDataCount

On return, the number of entries in the array returned in the *oLayoutDataArray* array.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The function `ATSUDirectGetLayoutDataArrayPtrFromLineRef` returns the data pointer specified by the `iDataSelector` parameter and referenced by the `iLineRef` parameter. You must call this function from within an `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998) callback function. You must only release the data pointer by calling the function `ATSUDirectReleaseLayoutDataArrayPtr` (page 1874). When you call this function, it signals ATSUI that you are done with the data and that ATSUI can merge your modifications with the font's data. If you do not properly free the data by calling the function `ATSUDirectReleaseLayoutDataArrayPtr`, a memory leak may result.

The data you obtain is the actual data used by ATSUI in its layout process; it is not a copy. This function is very efficient because ATSUI does not need to allocate memory and copy data. Furthermore, because you obtain a pointer to the data that ATSUI uses for its layout, any modifications you make to the data effect the final layout.

Many of the data arrays you can request are created by ATSUI only when necessary. If you plan to alter the data in an array, make sure you set the `iCreate` parameter to true. This ensures that the array is created. If an arrays are not created, ATSUI assumes all entries in the array are zero.

The pointer returned by this function is only valid within the context of the `ATSUDirectLayoutOperationOverrideProcPtr` callback function. You must not retain it for later use.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeDirectAccess.h`

ATSUDirectGetLayoutDataArrayPtrFromTextLayout

Obtains a copy of the glyph data specified by a direct-data selector and for a specific line of text in a text layout object.

```
OSStatus ATSUDirectGetLayoutDataArrayPtrFromTextLayout (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineOffset,
   ATSUDirectDataSelector iDataSelector,
   void *oLayoutDataArrayPtr[],
   ItemCount *oLayoutDataCount
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value that specifies the text layout object whose data you want to obtain.

iLineOffset
The edge offset that corresponds to the beginning of the line of text whose data you want to obtain.

iDataSelector
A direct-data selector constant that specifies the data you want to obtain. You can pass any of the constants described in "Direct Data Selectors" (page 2044).

oLayoutDataArrayPtr[]
On return, points to an array that contains the data specified by the `iDataSelector` parameter. The data is for the line of text referenced by the `iLineOffset` parameter. If an array for the specified data does not exist, ATSUI returns `NULL`. You can pass `NULL` if you only want to obtain the number of entries in the array in the *oLayoutDataArray* array.

oLayoutDataCount
On return, the number of entries in the array `oLayoutDataArray`.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The function `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` returns a pointer to the data specified by `iDataSelector` and referenced by `iTextLayout` for the line starting at `iLineOffset`. You must not call this function from within an `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998)callback function.

You should only release the data pointer by calling the function `ATSUDirectReleaseLayoutDataArrayPtr`. When you call this function, it signals ATSUI that you are done with the data and that ATSUI can merge your modifications with the font's data. If you do not properly free the data by calling the function `ATSUDirectReleaseLayoutDataArrayPtr`, a memory leak may result.

The data you obtain is a copy of the data ATSUI uses for its layout processes. This means the following:

- Obtaining data through a copy operation takes more time than obtaining the actual data. This function returns in order-n time instead of in a constant time.
- Changing any of the data values has no effect on the layout.

Before you use this function, you should consider using the function`ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) with the `kATSULayoutOperationPostLayoutAdjustment` selector.

If you use the function `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` to obtain the `ATSUStyleSettingRef` array, the structures referenced by each element of the array are invalid after you call the function `ATSUDirectReleaseLayoutDataArrayPtr` to release the array. If want to retain one or more of the elements in the `ATSUStyleSettingRef` array for later use, you must not call the function `ATSUDirectReleaseLayoutDataArrayPtr` until all operations that use the elements in the `ATSUStyleSettingRef` in the array are complete. The elements in the `ATSUStyleSettingRef` array are valid only within the context of the callback from which they were obtained

Many of the requested data arrays are created by ATSUI only when necessary. This means that it's possible for the function `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` to return a `NULL` pointer and a count of `0`. If this is case and if the function does not return an error, the array doesn't exist. You should interpret this result to mean that all values in the array are `0`.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDirectAccess.h`

ATSUDirectReleaseLayoutDataArrayPtr

Releases a pointer to a direct-data array.

```
OSStatus ATSUDirectReleaseLayoutDataArrayPtr (
   ATSULineRef iLineRef,
   ATSUDirectDataSelector iDataSelector,
   void *iLayoutDataArrayPtr[]
);
```

Parameters

`iLineRef`

An `ATSULineRef` value that specifies the line of text whose data is pointed to by the `iLayoutDataArrayPtr` parameter. Pass `NULL` if you did not obtain the layout data array pointer using a `lineRef`.

iDataSelector

A direct-data selector constant that specifies the data pointed to by the `iLayoutDataArrayPtr` parameter. You can pass any of the constants described in "Direct Data Selectors" (page 2044).

iLayoutDataArrayPtr[]

A pointer to the layout data array of which you want to dispose.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You must call the function `ATSUDirectReleaseLayoutDataArrayPtr` when you no longer need the direct-data pointer you obtained from the `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) functions. You must dispose of the pointer to inform ATSUI you no longer need the data and to allow for ATSUI to make any internal adjustments prior to completing the layout process.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeDirectAccess.h`

ATSUDisposeFontFallbacks

Disposes of the memory associated with a font fallback object.

```
OSStatus ATSUDisposeFontFallbacks (
   ATSUFontFallbacks iFontFallbacks
);
```

Parameters

iFontFallbacks

An `ATSUFontFallbacks` value specifying the font fallback object to dispose. See the `ATSUFontFallbacks` data type.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUDisposeFontFallbacks` function frees the memory associated with the specified font fallback object and its internal structures.

For best performance, once you create a font fallback object, you should keep it and use it as often as needed. You should dispose of the font fallback object only when it is no longer needed in your application.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUDisposeStyle

Disposes of the memory associated with a style object.

```
OSStatus ATSUDisposeStyle (
   ATSUStyle iStyle
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object to dispose of.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUDisposeStyle` function frees the memory associated with the specified style object and its internal structures, including style run attributes. It does not dispose of the memory pointed to by application-defined style run attributes or reference constants. You are responsible for doing so.

You should call this function after calling the function `ATSUDisposeTextLayout` (page 1876) to dispose of any text layout objects associated with the style object.

For best performance, once you create a style object, you should keep it and use it as often as needed. You should dispose of the style object only when it is no longer needed in your application.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSUnicodeObjects.h`

ATSUDisposeTextLayout

Disposes of the memory associated with a text layout object.

```
OSStatus ATSUDisposeTextLayout (
   ATSUTextLayout iTextLayout
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to dispose of.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUDisposeTextLayout` function frees the memory associated with the specified text layout object and its internal structures, including line and layout control attributes, style runs, and soft line breaks. `ATSUDisposeTextLayout` does not dispose of any memory that may be allocated for reference constants or style objects associated with the text layout object. You are responsible for doing so.

For best performance, text layout objects are readily reusable and should be cached for later use, if possible. You can reuse a text layout object even if the text associated with it is altered. Call the functions `ATSUSetTextPointerLocation` (page 1965), `ATSUTextDeleted` (page 1969), or `ATSUTextInserted` (page 1970) to manage the altered text, rather than disposing of the text layout object and creating a new one.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeObjects.h`

ATSUDrawGlyphInfo

Draws glyphs at the specified location, based on style and layout information specified for each glyph. (Deprecated in Mac OS X v10.3. Use functions from "Accessing Glyph Data" (page 1842) instead.)

Not recommended.

```
OSStatus ATSUDrawGlyphInfo (
   ATSUGlyphInfoArray *iGlyphInfoArray,
   Float32Point iLocation
);
```

Parameters

iGlyphInfoArray

A pointer to an `ATSUGlyphInfoArray` structure containing the glyph information to draw. You can obtain an `ATSUGlyphInfoArray` structure from the function `ATSUGetGlyphInfo` (page 1906).

iLocation

A `Float32Point` data structure that contains the x and y coordinates at which to draw the glyph(s). Each coordinate in the `Float32Point` data structure is a `Float32` value.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You must use `ATSUDrawGlyphInfo` to draw glyphs if you have previously called the function `ATSUGetGlyphInfo` (page 1906), and you have modified the glyph information. However, if you want to modify the glyph information you should use the functions `ATSUGlyphGetQuadraticPaths` (page 1928) or `ATSUGlyphGetCubicPaths` (page 1925) instead of calling the function `ATSUGetGlyphInfo`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`ATSUnicodeGlyphs.h`

ATSUDrawText

Renders a range of text at a specified location in a QuickDraw graphics port or Quartz graphics context.

```
OSStatus ATSUDrawText (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineOffset,
   UniCharCount iLineLength,
   ATSUTextMeasurement iLocationX,
   ATSUTextMeasurement iLocationY
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object for which to render text.

iLineOffset

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the range to render. The function `ATSUDrawText` renders text to the first soft line break it encounters. If the range of text spans multiple lines, you should call `ATSUDrawText` for each line, passing the offset corresponding to the beginning of the new line to draw with each call. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iLineLength` parameter.

iLineLength

A `UniCharCount` value specifying the length of the text range to render. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`. Keep in mind that the function `ATSUDrawText` renders text one line at a time. If the range of text spans multiple lines, you must call `ATSUDrawText` for each line.

iLocationX

An `ATSUTextMeasurement` value specifying the x-coordinate of the origin (in either the current QuickDraw graphics port or in a Quartz graphics context) of the line containing the text range to render. Note that the `ATSUTextMeasurement` type is defined as a `Fixed` value, so you must ensure that your coordinates are converted to `Fixed` values before passing them to this function. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to draw relative to the current pen location in the current graphics port.

iLocationY

An `ATSUTextMeasurement` value specifying the y-coordinate of the origin (in either the current graphics port or Quartz graphics context) of the line containing the text range to render. Note that the `ATSUTextMeasurement` type is defined as a `Fixed` value, so you must ensure that your coordinates are converted to `Fixed` values before passing them to this function. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to draw relative to the current pen location in the current graphics port.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUDrawText` function renders a range of text at a specified location in a QuickDraw graphics port or Quartz graphics context. This function renders text to the first soft line break it encounters. If you draw into a QuickDraw graphics port you get the best performance by using a bit depth of 16 bits. If you use bit depths of 1, 4, or 8, your application incurs a performance penalty.

You typically call the `ATSUDrawText` function every time you need to draw or redraw unhighlighted text. To draw highlighted text, call the function `ATSUHighlightText` (page 1931).

ATSUDrawText uses the transfer mode and resolution that are set in the graphics port or graphics context. If you explicitly set in the style object, then text color is taken from the style object, and the value in the graphics port/context is ignored. If the text color was not explicitly set in the style object, ATSUDrawText uses the graphics port/context setting.

ATSUDrawText examines the text layout object to ensure that each of the characters in the range is assigned to a style run. If there are gaps between style runs, ATSUI assigns the characters in the gap to the style run that precedes (in storage order) the gap. If there is no style run at the beginning of the text range, ATSUI assigns these characters to the first style run it finds. If there is no style run at the end of the text range, ATSUI assigns the remaining characters to the last style run it finds.

If you want to draw a range of text that spans multiple lines, you should call ATSUDrawText for each line of text to draw, even if all the lines are in the same text layout object. You should adjust the iLineOffset parameter to reflect the beginning of each line to be drawn.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeDrawing.h

ATSUFindFontFromName

Obtains an ATSUI font ID for the first entry in a name table that matches the specified name string, name code, platform, script, and/or language.

```
OSStatus ATSUFindFontFromName (
   const void *iName,
   ByteCount iNameLength,
   FontNameCode iFontNameCode,
   FontPlatformCode iFontNamePlatform,
   FontScriptCode iFontNameScript,
   FontLanguageCode iFontNameLanguage,
   ATSUFontID *oFontID
);
```

Parameters

iName

A string that specifies the font name whose ATSUI font ID you want to obtain. The string that you pass must be appropriate for the value you pass in the iFontNameCode parameter. For example, if the iFontNameCode parameter is kFontPostscriptName, then you would supply a string that specifies the PostScript name of the font.

iNameLength

A ByteCount value specifying the length of the font name string provided in the *iName* parameter.

iFontNameCode

The FontNameCode value of the font name for which to obtain an ATSUI font ID. The FontNameCode is a UInt32 data type, and it is defined in the SFNTTypes.h header file. You can supply any of the following constants, kFontCopyrightName, kFontFamilyName, kFontStyleName, kFontUniqueName, kFontFullName, kFontVersionName, kFontPostscriptName, kFontTrademarkName, kFontManufacturerName, kFontDesignerName, kFontDescriptionName, kFontVendorURLName, kFontDesignerURLName, kFontLicenseDescriptionName,or kFontLicenseInfoURLName.

iFontNamePlatform

A FontPlatformCode value specifying the encoding of the font name, for example, kFontUnicodePlatform (for UTF-16), kFontMacintoshPlatform, kFontReservedPlatform, kFontMicrosoftPlatform, or kFontCustomPlatform. If you pass the kFontNoPlatformCode constant, ATSUFindFontFromName produces the first font in the name table matching the other specified parameters. See the SFNTTypes.h header file for a definition of the FontPlatformCode type and a list of possible values.

iFontNameScript

A FontScriptCode value specifying the script code of the font name, for example, kFontRomanScript. Pass kFontNoScriptCode if you supplied the kFontUnicodePlatform constant for the iFontNamePlatform parameter. If you pass the kFontNoScriptCode constant, ATSUFindFontFromName produces the first font in the name table matching the other specified parameters. See the SFNTTypes.h header file for a definition of the FontScriptCode type and a list of possible values.

iFontNameLanguage

A FontLanguageCode value specifying the language of the font name, for example, kFontNorwegianLanguage. Pass kFontNoLanguageCode if you supplied the kFontUnicodePlatform constant for the iFontNamePlatform parameter. If you pass the kFontNoLanguageCode constant, ATSUFindFontFromName produces the first font in the name table matching the other specified parameters. See the SFNTTypes.h header file for a definition of the FontLanguageCode type and a list of possible values.

oFontID

On return, points to the unique identifier for the specified font that matches the specified name string, name code, platform, script, and/or language. Note that because Apple Type Services assigns ATSUFontID values systemwide at runtime, font IDs can change across system restarts.

Return Value

A result code. If no installed font matches the specified parameters, ATSUFindFontFromName produces the constant kATSUInvalidFontID and returns the result code kATSUInvalidFontErr. See "ATSUI Result Codes" (page 2068).

Discussion

The ATSUFindFontFromName function obtains an ATSUI font ID for the first font that matches the specified name string, name code, platform, script, and/or language. Because ATSUI cannot guarantee the uniqueness of names among installed fonts, ATSUFindFontFromName does not necessarily find the only font ID that matches these parameters. As a result, you may want to create a more sophisticated name-matching algorithm or guarantee the uniqueness of names among installed fonts.

To find a name string and index value for the first font in a name table that matches an ATSUI font ID and the specified font parameters, call the function ATSUFindFontName (page 1880).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

ATSUnicodeFonts.h

ATSUFindFontName

Obtains a name string and index value for the first font in a name table that matches the specified ATSUI font ID, name code, platform, script, and/or language.

```
OSStatus ATSUFindFontName (
   ATSUFontID iFontID,
   FontNameCode iFontNameCode,
   FontPlatformCode iFontNamePlatform,
   FontScriptCode iFontNameScript,
   FontLanguageCode iFontNameLanguage,
   ByteCount iMaximumNameLength,
   Ptr oName,
   ByteCount *oActualNameLength,
   ItemCount *oFontNameIndex
);
```

Parameters

iFontID

The `ATSUFontID` value of the font for which to obtain a name string. Note that because Apple Type Services assigns `ATSUFontID` values systemwide at runtime, font IDs can change across system restarts.

iFontNameCode

The `FontNameCode` value of the font for which to obtain a name string. The `FontNameCode` is a `UInt32` data type, and it is defined in the `SFNTTypes.h` header file.

iFontNamePlatform

A `FontPlatformCode` value specifying the encoding of the font, for example, `kFontUnicodePlatform`, `kFontMacintoshPlatform`, `kFontReservedPlatform`, `kFontMicrosoftPlatform`, or `kFontCustomPlatform`. If you pass the `kFontNoPlatformCode` constant, `ATSUFindFontName` produces the first font in the name table matching the other specified parameters. See the `SFNTTypes.h` header file for a definition of the `FontPlatformCode` type and a list of possible values.

iFontNameScript

A `FontScriptCode` value specifying the script code of the font, for example, `kFontRomanScript`. If you pass the `kFontNoScriptCode` constant, `ATSUFindFontName` produces the first font in the name table matching the other specified parameters. See the `SFNTTypes.h` header file for a definition of the `FontScriptCode` type and a list of possible values.

iFontNameLanguage

A `FontLanguageCode` value specifying the language of the font, for example, `kFontNorwegianLanguage`. If you pass the `kFontNoLanguageCode` constant, `ATSUFindFontName` produces the first font in the name table matching the other specified parameters. See the `SFNTTypes.h` header file for a definition of the `FontLanguageCode` type and a list of possible values.

iMaximumNameLength

A `ByteCount` value specifying the maximum length of the font name to obtain. Typically, this is equivalent to the size of the buffer that you have allocated in the `oName` parameter. To determine this length, see the Discussion.

oName

A pointer to a buffer. On return, the buffer contains the name string of the first font in the font name table matching your specified parameters. If the buffer you allocate is not large enough, `ATSUFindFontName` produces a partial string.

oActualNameLength

A pointer to a `ByteCount` value. On return, the value specifies the actual length of the complete name string. This may be greater than the value passed in the `iMaximumNameLength` parameter. You should check this value to ensure that you have allocated sufficient memory and therefore obtained the complete name string for the font.

oFontNameIndex

A pointer to an ItemCount value. On return, the value provides a 0-based index to the font name in the font name table.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The ATSUFindFontName function obtains a name string and index value for the first font in a name table that matches the specified ATSUI font ID, name code, platform, script, and/or language.

Typically you use the ATSUFindFontName function by calling it twice, as follows:

1. Pass NULL for the *oName* and *oFontNameIndex* parameters, 0 for the *iMaximumNameLength* parameter, and valid values for the other parameters. ATSUFindFontName returns the length of the font name string in the oActualNameLength parameter.

2. Allocate enough space for a buffer of the returned size, then call the function again, passing a valid pointer to the buffer in the oName parameter. On return, the buffer contains the font name string.

To obtain an ATSUI font ID for the first font in a name table that matches the specified name string, name code, platform, script, and/or language, call the function ATSUFindFontFromName (page 1879). To obtain the font name string, name code, platform, script, and language for the font that matches an ATSUI font ID and name table index, call the function ATSUGetIndFontName (page 1908).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeFonts.h

ATSUFlattenStyleRunsToStream

Flattens ATSUI style-run data so that it can be saved to disk or passed (through the pasteboard) to another application.

```
OSStatus ATSUFlattenStyleRunsToStream (
   ATSUFlattenedDataStreamFormat iStreamFormat,
   ATSUFlattenStyleRunOptions iFlattenOptions,
   ItemCount iNumberOfRunInfo,
   const ATSUStyleRunInfo iRunInfoArray[],
   ItemCount iNumberOfStyleObjects,
   const ATSUStyle iStyleArray[],
   ByteCount iStreamBufferSize,
   void *oStreamBuffer,
   ByteCount *oActualStreamBufferSize
);
```

Parameters

iStreamFormat

The format of the flattened data. There is only one format supported at this time, 'ustl' so you must pass the constant kATSUDataStreamUnicodeStyledText.

iFlattenOptions

The options you want to use to flatten the data. There are no options supported at this time, so you must pass the constant `kATSUFlattenOptionsNoOptionsMask`.

iNumberOfRunInfo

The number of style run information structures passed in the `iRunInfoArray` parameter. If you pass `0`, ATSUI assumes there is only one style for the entire text block passed in the `oStreamBuffer` parameter. The flattened data format passed to the `iStreamFormat` parameter must support the use of one style.

iRunInfoArray[]

An array of `ATSUStyleRunInfo` structures that describes the style runs to be flattened. This array must contain `iNumberOfRunInfo` entries. An `ATSUStyleRunInfo` structure contains an index into an array of unique ATSUI style objects (`ATSUStyle`) and the length of the run to which the style object applies. Each index in the `ATSUStyleRunInfo` structure must reference a valid `ATSUStyle` object passed in the `iStyleArray` parameter. You can pass `NULL`, only if `iNumberOfRunInfo` is set to zero.

iNumberOfStyleObjects

The number of `ATSUStyle` objects in the array passed to the `iStyleArray` parameter. You must pass a value that is greater than `0`.

iStyleArray[]

An array of `ATSUStyle` objects to be flattened. You cannot pass `NULL`.

iStreamBufferSize

The size of the stream buffer, pointed to by the `oStreamBuffer` parameter. You can pass `0` only if the `iStreamBufferSize` parameter is set to `NULL`. If you are uncertain of the size of the array, see the Discussion.

oStreamBuffer

On input, a pointer to the data you want to flatten. On return, points to the flattened data. If you pass `NULL` for this parameter, no data is flattened. Instead, the size of the buffer is calculated by ATSUI and returned in `oActualStreamSize` parameter. See the Discussion for more details. You are responsible for allocating the text buffer passed in the `oStreamBuffer` parameter.

oActualStreamBufferSize

On return, the size of the data written to the `oStreamBuffer` parameter. You can pass `NULL` only if the `oStreamBuffer` parameter is not `NULL`.

Return Value

A result code. See "ATSUI Result Codes" (page 2068). This function can also return `paramErr` if you pass invalid values for any of the parameters.

Discussion

The function `ATSUFlattenStyleRunsToStream` takes an array of `ATSUStyle` objects and style run information and flattens the data to the specified format. The style runs must all reference the same block of Unicode text (usually passed separately as text in the `'utxt'` format). The style runs must also be in ascending order relative to the text in the text block.

Typically you use the function `ATSUFlattenStyleRunsFromStream` by calling it twice, as follows:

1. Provide appropriate values for the `iStreamFormat`, `iFlattenOptions`, `iNumberOfRunInfo`, `iRunInfoArray`, `iNumberOfStyleObjects`, and `iStyleArray` parameters. Set `iStreamBufferSize` to `0`, `oStreamBuffer` to `NULL`, and pass a valid reference to a `ByteCount` variable in the `oActualStreamBufferSize` parameter. Call the function `ATSUFlattenStyleRunsToStream`. On return, `oActualStreamBufferSize` points to the size needed for the buffer.

2. Allocate an appropriately-sized buffer for the `oStreamBuffer` parameter and then call the function `ATSUFlattenStyleRunsToStream` a second time.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFlattening.h`

ATSUFONDtoFontID

Finds the ATSUI font ID that corresponds to a font family number, if one exists. (**Deprecated.** There is no replacement because FONDs are a QuickDraw concept and QuickDraw is deprecated.)

Not recommended.

```
OSStatus ATSUFONDtoFontID (
   short iFONDNumber,
   Style iFONDStyle,
   ATSUFontID *oFontID
);
```

Parameters

iFONDNumber
The font family number of the ATSUI-compatible font for which to obtain an ATSUI font ID.

iFONDStyle
The font family style of the font, if any. Style identifiers exist only for fonts that split a font family into subgroups.

oFontID
A pointer to a `ATSUFontID` value. On return, the value provides a unique identifier for the specified font family number and style.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The function `ATSUFONDtoFontID` is not recommended for use. Instead, use the Font Manager functions that translate font family numbers to `FMFont` values, which are equivalent to `ATSUFontID` values. Font family numbers were used by QuickDraw to represent fonts to the Font Manager. Some of these fonts, even if compatible with ATSUI, may not have font IDs.

Note that Apple Type Services assigns `ATSUFontID` values systemwide at runtime. As a result, these font IDs can change when the system is restarted.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUFontCount

Obtains the number of ATSUI-compatible fonts installed on a user's system.

```
OSStatus ATSUFontCount (
   ItemCount *oFontCount
);
```

Parameters

oFontCount

A pointer to an `ItemCount` value. On return, the value specifies the current number of ATSUI-compatible fonts installed on the user's system.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUFontCount` function obtains the number of fonts on a user's system that are compatible with ATSUI. Incompatible fonts include those that cannot be used to represent Unicode, the missing-character glyph font, and fonts whose names begin with a period or a percent sign. You can use the count produced in the `oFontCount` parameter to determine the amount of memory to allocate for the `oFontIDs` array in the function `ATSUGetFontIDs` (page 1899).

It is important to note that the set of installed ATSUI-compatible fonts may change while your application is running. In Mac OS X, the set of installed fonts may change at any time. Although in Mac OS 9, fonts cannot be removed from the Fonts folder while an application other than the Finder is running, they can be removed from other locations, and it is possible for fonts to be added.

Additionally, just because the number of fonts stays the same between two successive calls to `ATSUFontCount`, this does not mean that the font lists are the same. It is possible for a font to be added and another removed between two successive calls to `ATSUFontCount`, leaving the total number unchanged.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeFonts.h`

ATSUFontIDtoFOND

Finds the font family number that corresponds to an ATSUI font ID, if one exists. (**Deprecated.** There is no replacement because FONDs are a QuickDraw concept and QuickDraw is deprecated.)

Not recommended.

```
OSStatus ATSUFontIDtoFOND (
   ATSUFontID iFontID,
   short *oFONDNumber,
   Style *oFONDStyle
);
```

Parameters

iFontID

The `ATSUFontID` value of the font for which to obtain a font family number. Note that because Apple Type Services assigns `ATSUFontID` values systemwide at runtime, font IDs can change across system restarts.

oFONDNumber

A pointer to a signed sixteen-bit integer. On return, the value identifies the font family number corresponding to the specified ATSUI font ID.

oFONDStyle

A pointer to a `Style` value. On return, the value identifies the font family style of the font, if any. Style identifiers exist only for fonts that split a font family into subgroups.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The function `ATSUFontIDtoFOND` is not recommended for use. Instead, use the Font Manager functions that translate `FMFont` values, which are equivalent to `ATSUFontID` values, to font family numbers. Font family numbers were used by QuickDraw to represent fonts to the Font Manager. Some of these fonts, even if compatible with ATSUI, may not have font IDs.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeFonts.h`

ATSUGetAllAttributes

Obtains an array of style attribute tags and value sizes for a style object.

```
OSStatus ATSUGetAllAttributes (
   ATSUStyle iStyle,
   ATSUAttributeInfo oAttributeInfoArray[],
   ItemCount iTagValuePairArraySize,
   ItemCount *oTagValuePairCount
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object to examine.

oAttributeInfoArray

A pointer to memory you have allocated for an array of `ATSUAttributeInfo` values. On return, the array contains pairs of tags and value sizes for any of the object's style attributes that are not at default values. If you are uncertain of how much memory to allocate for this array, see the Discussion.

iTagValuePairArraySize

An ItemCount value specifying the maximum number of tag and value size pairs to obtain for the style object. Typically, this is equivalent to the number of ATSUAttributeInfo structures for which you have allocated memory in the oAttributeInfoArray parameter. To determine this value, see the Discussion.

oTagValuePairCount

A pointer to an ItemCount value. On return, the value specifies the actual number of ATSUAttributeInfo structures in the style object. This may be greater than the value you specified in the iTagValuePairArraySize parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The ATSUGetAllAttributes function obtains all nondefault style attribute tags and values sizes for a style object. You can pass a tag and value-size pair obtained from ATSUGetAllAttributes to the function ATSUGetAttribute (page 1892) to determine the corresponding attribute value.

Typically you use the function ATSUGetAllAttributes by calling it twice, as follows:

1. Pass a reference to the style object to examine in the iStyle parameter, a valid pointer to an ItemCount value in the *oTagValuePairCount* parameter, NULL for the oAttributeInfoArray parameter, and 0 for the *iTagValuePairArraySize* parameter. ATSUGetAllAttributes returns the size of the tag and value-size arrays in the oTagValuePairCount parameter.

2. Allocate enough space for an array of the returned size, then call the ATSUGetAllAttributes function again, passing a valid pointer in the oAttributeInfoArray parameter. On return, the pointer refers to an array of the style attribute tag and value-size pairs contained in the style object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeObjects.h

ATSUGetAllFontFeatures

Obtains the font features of a style object that are not at default settings.

```
OSStatus ATSUGetAllFontFeatures (
   ATSUStyle iStyle,
   ItemCount iMaximumFeatureCount,
   ATSUFontFeatureType oFeatureType[],
   ATSUFontFeatureSelector oFeatureSelector[],
   ItemCount *oActualFeatureCount
);
```

Parameters

iStyle

An ATSUStyle value specifying the style object to examine.

iMaximumFeatureCount

An ItemCount value specifying the maximum number of feature types and selectors to obtain for the style object. Typically, this is equivalent to the number of ATSUFontFeatureType and ATSUFontFeatureSelector values for which you have allocated memory in the oFeatureType and oFeatureSelector parameters, respectively. To determine this value, see the Discussion.

oFeatureType

A pointer to memory you have allocated for an array of ATSUFontFeatureType values. On return, the array contains constants identifying each type of font feature that is at a nondefault setting in the style object. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oFeatureSelector

A pointer to memory you have allocated for an array of ATSUFontFeatureSelector values. On return, the array contains constants identifying the feature selectors that are at nondefault settings in the style object. Each selector determines the setting for a corresponding feature type produced in the *oFeatureType* parameter. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oActualFeatureCount

A pointer to an ItemCount value. On return, the value specifies the actual number of font feature types and selectors in the style object. This may be greater than the value you specified in the iMaximumFeatureCount parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The ATSUGetAllFontFeatures function obtains all of a style object's font features that are not at default settings. Font features are grouped into categories called feature types, within which individual feature selectors define particular feature settings. The arrays produced by ATSUGetAllFontFeatures contain constants identifying the object's font types and their corresponding font selectors.

Typically you use the function ATSUGetAllFontFeatures by calling it twice, as follows:

1. Pass a reference to the style object to examine in the iStyle parameter, a valid pointer to an ItemCount value in the *oActualFeatureCount* parameter, NULL for the oFeatureType and oFeatureSelector parameters, and 0 for the *iMaximumFeatureCount* parameter. ATSUGetAllFontFeatures returns the size in the oActualFeatureCount parameter to use for the feature type and selector arrays.

2. Allocate enough space for arrays of the returned size, then call ATSUGetAllFontFeatures again, passing a pointer to the arrays in the oFeatureType and oFeatureSelector parameters. On return, the arrays contain the font feature types and selectors, respectively, for the style object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeFonts.h

ATSUGetAllFontVariations

Obtains a style object's font variation values that are not at default settings.

```
OSStatus ATSUGetAllFontVariations (
   ATSUStyle iStyle,
   ItemCount iVariationCount,
   ATSUFontVariationAxis oVariationAxes[],
   ATSUFontVariationValue oFontVariationValues[],
   ItemCount *oActualVariationCount
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object to examine.

iVariationCount

An `ItemCount` value specifying the maximum number of font variation values to obtain for the style object. Typically, this is equivalent to the number of `ATSUFontVariationAxis` and `ATSUFontVariationValue` values for which you have allocated memory in the `oVariationAxes` and `oFontVariationValues` parameters, respectively. To determine this value, see the Discussion.

oVariationAxes

A pointer to memory you have allocated for an array of `ATSUFontVariationAxis` values. On return, the array contains tags identifying those font variation axes in the style object that are not at default values. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oFontVariationValues

A pointer to memory you have allocated for an array of `ATSUFontVariationValue` values. On return, the array contains the current font variation values for the font variation axes produced in the `oVariationAxes` array. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oActualVariationCount

A pointer to an `ItemCount` value. On return, the value specifies the actual number of nondefault font variation values in the style object. This may be greater than the value you passed in the `iVariationCount` parameter.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUGetAllFontVariations` function obtains all of a style object's font variation axes that are not at default settings, as well as the current values for the axes.

Typically you use the function `ATSUGetAllFontVariations` by calling it twice, as follows:

1. Pass a reference to the style object to examine in the `iStyle` parameter, a pointer to an `ItemCount` value in the *oActualVariationCount* parameter, `NULL` for the `oVariationAxes` and `oFontVariationValues` parameters, and 0 for the *iVariationCount* parameter. `ATSUGetAllFontVariations` returns the size to use for the variation axes and value arrays in the `oActualVariationCount` parameter.

2. Allocate enough space for arrays of the returned size, then call `ATSUGetAllFontVariations` again, passing a pointer to the arrays in the `oVariationAxes` and `oFontVariationValues` parameters. On return, the arrays contain the font variation axes and their corresponding values, respectively, for the style object.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
ATSUnicodeFonts.h

ATSUGetAllLayoutControls

Obtains an array of layout control attribute tags and value sizes for a text layout object.

```
OSStatus ATSUGetAllLayoutControls (
   ATSUTextLayout iTextLayout,
   ATSUAttributeInfo oAttributeInfoArray[],
   ItemCount iTagValuePairArraySize,
   ItemCount *oTagValuePairCount
);
```

Parameters

iTextLayout

An ATSUTextLayout value specifying the text layout object to examine.

oAttributeInfoArray

A pointer to memory you have allocated for an array of ATSUAttributeInfo values. On return, the array contains pairs of tags and value sizes for the object's layout control attributes that are not at default values. If you are uncertain of how much memory to allocate for this array, see the Discussion.

iTagValuePairArraySize

An ItemCount value specifying the maximum number of tag and value size pairs to obtain for the text layout object. Typically, this is equivalent to the number of ATSUAttributeInfo structures for which you have allocated memory in the oAttributeInfoArray parameter. To determine this value, see the Discussion.

oTagValuePairCount

A pointer to an ItemCount value. On return, the value specifies the actual number of ATSUAttributeInfo structures in the text layout object. This may be greater than the value you specified in the iTagValuePairArraySize parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The ATSUGetAllLayoutControls function obtains all nondefault layout control attribute tags and their values sizes for a text layout object. You can pass a tag and value size pair obtained from ATSUGetAllLayoutControls to the function ATSUGetLayoutControl (page 1912) to determine the corresponding attribute value.

Typically you use the function ATSUGetAllLayoutControls by calling it twice, as follows:

1. Pass a reference to the text layout object to examine in the iTextLayout parameter, NULL for the oAttributeInfoArray parameter, a pointer to an ItemCount value in the *oTagValuePairCount* parameter, and 0 for the *iTagValuePairArraySize* parameter. ATSUGetAllLayoutControls returns the size of the tag and value size arrays in the oTagValuePairCount parameter.

2. Allocate enough space for an array of the returned size, then call the ATSUGetAllLayoutControls function again, passing a valid pointer in the oAttributeInfoArray parameter. On return, the pointer refers to an array of the layout control attribute tag and value size pairs contained in the text layout object.

To obtain the nondefault line control attribute tags and value sizes for a text layout object, call the function `ATSUGetAllLineControls` (page 1891).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetAllLineControls

Obtains an array of line control attribute tags and value sizes for a line in a text layout object.

```
OSStatus ATSUGetAllLineControls (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart,
   ATSUAttributeInfo oAttributeInfoArray[],
   ItemCount iTagValuePairArraySize,
   ItemCount *oTagValuePairCount
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object to examine.

iLineStart
A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the line for which to obtain line control attribute values.

oAttributeInfoArray
A pointer to memory you have allocated for an array of `ATSUAttributeInfo` values. On return, the array contains pairs of tags and value sizes for the object's line control attributes that are not at default values. If you are uncertain of how much memory to allocate for this array, see the Discussion.

iTagValuePairArraySize
An `ItemCount` value specifying the maximum number of tag and value size pairs to obtain for the line. Typically, this is equivalent to the number of `ATSUAttributeInfo` structures for which you have allocated memory in the `oAttributeInfoArray` parameter. To determine this value, see the Discussion.

oTagValuePairCount
A pointer to an `ItemCount` value. On return, the value specifies the actual number of `ATSUAttributeInfo` structures in the line. This may be greater than the value you specified in the `iTagValuePairArraySize` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetAllLineControls` function obtains all nondefault line control attribute tags and their values sizes for a line in a text layout object. You can pass a tag and value size pair obtained from `ATSUGetAllLineControls` to the function `ATSUGetLineControl` (page 1913) to determine the corresponding attribute value.

Typically you use the function `ATSUGetAllLineControls` by calling it twice, as follows:

1. Pass a reference to the text layout object to examine in the `iTextLayout` parameter, the appropriate `UniCharArrayOffset` value in the *`iLineStart`* parameter, `NULL` for the `oAttributeInfoArray` parameter, a pointer to an `ItemCount` value in the *`oTagValuePairCount`* parameter, and 0 for the *`iTagValuePairArraySize`* parameter. `ATSUGetAllLineControls` returns the size of the tag and value size arrays in the `oTagValuePairCount` parameter.

2. Allocate enough space for an array of the returned size, then call the `ATSUGetAllLineControls` function again, passing a valid pointer in the `oAttributeInfoArray` parameter. On return, the pointer refers to an array of the line control attribute tag and value size pairs contained in the specified line.

To obtain the nondefault layout control attribute tags and value sizes for a text layout object, call the function `ATSUGetAllLayoutControls` (page 1890).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetAttribute

Obtains a style attribute value for a style object.

```
OSStatus ATSUGetAttribute (
   ATSUStyle iStyle,
   ATSUAttributeTag iTag,
   ByteCount iExpectedValueSize,
   ATSUAttributeValuePtr oValue,
   ByteCount *oActualValueSize
);
```

Parameters

`iStyle`
An `ATSUStyle` value specifying the style object for which to obtain an attribute value.

`iTag`
An `ATSUAttributeTag` constant identifying the attribute value to obtain. See "Attribute Tags" (page 2030) for a description of the Apple-defined style attribute tag constants.

`iExpectedValueSize`
The expected size (in bytes) of the value to obtain. To determine the size of an application-defined style attribute value, see the Discussion.

`oValue`
An `ATSUAttributeValuePtr` value, identifying the memory you have allocated for the attribute value. If you are uncertain of how much memory to allocate, see the Discussion. On return, *`oValue`* contains a valid pointer to the actual attribute value.

`oActualValueSize`
A pointer to a `ByteCount` value. On return, the value contains the actual size (in bytes) of the attribute value. You should examine this parameter if you are unsure of the size of the attribute value being obtained, as in the case of custom style run attributes.

Return Value
A result code. See "ATSUI Result Codes" (page 2068). Note that if the attribute value you want to obtain is not set, `ATSUGetAttribute` produces the default value in the `oValue` parameter and returns the result code `kATSUNotSetErr`.

Discussion
The `ATSUGetAttribute` function obtains the value of a specified style attribute for a given style object.

Before calling `ATSUGetAttribute`, you should call the function `ATSUGetAllAttributes` (page 1886) to obtain an array of nondefault style attribute tags and value sizes for the style object. You can then pass `ATSUGetAttribute` the tag and value size for the attribute value to obtain.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetContinuousAttributes

Obtains the style attribute values that are continuous over a given text range.

```
OSStatus ATSUGetContinuousAttributes (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOffset,
   UniCharCount iLength,
   ATSUStyle oStyle
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object to examine.

iOffset
A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the text range to examine. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iLength` parameter.

iLength
A `UniCharCount` value specifying the length of the text range to examine. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

oStyle
An `ATSUStyle` value. On return, the style object contains those attributes that are the same for the entire text range specified by the `iOffset` and `iLength` parameters.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetContinuousAttributes` function examines the specified text range to obtain the style attribute values (including those at default values) that remain consistent for the entire text range. You should call `ATSUGetContinuousAttributes` to determine the style information that remains constant over text that has been selected by the user.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetFontFallbacks

Obtains the global font list and search order that ATSUI uses when a font does not have the glyph needed to image a character. (Deprecated in Mac OS X v10.3. Use font fallback objects instead.)

Not recommended.

```
OSStatus ATSUGetFontFallbacks (
   ItemCount iMaxFontFallbacksCount,
   ATSUFontID oFontIDs[],
   ATSUFontFallbackMethod *oFontFallbackMethod,
   ItemCount *oActualFallbacksCount
);
```

Parameters

iMaxFontFallbacksCount
An `ItemCount` value specifying the maximum number of fonts that you want to obtain. Typically, this is equivalent to the size of the array allocated in the `oFontIDs` parameter. To determine this value, see the Discussion.

oFontIDs
A pointer to memory you have allocated for an array of `ATSUFontID` values. If you are uncertain of how much memory to allocate, see the Discussion. On return, the array contains font IDs identifying the fonts ATSUI searches when seeking a substitute font.

oFontFallbackMethod
A pointer to an `ATSUFontFallbackMethod` value. On return, the value identifies the order in which ATSUI searches fonts. See "Font Fallback Methods" (page 2048) for a description of possible values.

oActualFallbacksCount
A pointer to an `ItemCount` value. On return, the value specifies the actual number of fonts that ATSUI searches. This value may be greater than that passed in the `iMaxFontFallbacksCount` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not use this function because it operates on a global scope and may not be available in future versions of ATSUI. You should instead use the function `ATSUGetObjFontFallbacks` (page 1914) with a font fallback object that has been associated with a text layout object. See *Inside Mac OS X: Rendering Unicode Text With ATSUI* for step-by-step instructions on creating a font fallback object and associating it with a text layout object.

Special Considerations

Global font fallback settings can be changed by any ATSUI client, so they can be changed unexpectedly. The only way to ensure that ATSUI uses your preferred font fallback settings for your text is to create a font fallback object and associated it with a text layout object. See the Discussion for more details.

Version Notes

Available beginning with ATSUI 1.1.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUGetFontFeature

Obtains the font feature corresponding to an index into an array of font features for a style object.

```
OSStatus ATSUGetFontFeature (
   ATSUStyle iStyle,
   ItemCount iFeatureIndex,
   ATSUFontFeatureType *oFeatureType,
   ATSUFontFeatureSelector *oFeatureSelector
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object to examine.

iFeatureIndex

An `ItemCount` value specifying an index into the array of font features for the style object. This index identifies the font feature to examine. Because this index is zero-based, you must pass a value between 0 and one less than the value produced in the `oActualFeatureCount` parameter of the function `ATSUGetAllFontFeatures` (page 1887).

oFeatureType

A pointer to memory you have allocated for an `ATSUFontFeatureType` value. On return, the value identifies the font feature type corresponding to the index passed in the `iFeatureIndex` parameter.

oFeatureSelector

A pointer to memory you have allocated for an `ATSUFontFeatureSelector` value. On return, the value identifies the font feature selector that corresponds to the feature type produced in the *oFeatureType* parameter.

Return Value

A result code. Note that if the index specifies a font feature that is not set, `ATSUGetFontFeature` produces the font-specified default value for the feature and returns the result code `kATSUNotSetErr`. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetFontFeature` function obtains the setting for a specified font feature in a style object. You might typically call `ATSUGetFontFeature` if you need to obtain one previously set feature after another within your program's processing loop. To obtain all previously set font features for a given style object, you can call the function `ATSUGetAllFontFeatures` (page 1887).

Before calling `ATSUGetFontFeature`, you should call the function `ATSUGetAllFontFeatures` (page 1887) to obtain a count of the font features that are set in the style object. You can then pass the index for the feature whose setting you want to obtain in the `iTag` and `iMaximumValueSize` parameters of `ATSUGetFontFeature`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetFontFeatureNameCode

Obtains the name code for a font's feature type or selector that matches an ASTUI font ID, feature type, and feature selector.

```
OSStatus ATSUGetFontFeatureNameCode (
   ATSUFontID iFontID,
   ATSUFontFeatureType iType,
   ATSUFontFeatureSelector iSelector,
   FontNameCode *oNameCode
);
```

Parameters

iFont
The `ATSUFontID` value of the font for which to obtain the name code for a feature type or selector. Note that because Apple Type Services assigns `ATSUFontID` values systemwide at runtime, font IDs can change across system restarts.

iType
An `ATSUFontFeatureType` constant identifying a valid feature type. To obtain the valid feature types for a font, call the function `ATSUGetFontFeatureTypes` (page 1898).

iSelector
An `ATSUFontFeatureSelector` constant identifying a valid feature selector that corresponds to the feature type passed in the `iType` parameter. If you pass the constant `kATSUNoSelector`, the name code produced by `ATSUGetFontFeatureNameCode` is that of the feature type, not the feature selector. To obtain the valid feature selectors for a font, call the function`ATSUGetFontFeatureSelectors` (page 1897).

oNameCode
A pointer to a `FontNameCode` value. On return, the value contains the name code for the font feature selector or type. The `FontNameCode` is a `UInt32` data type, and it is defined in the `SFNTTypes.h` header file.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetFontFeatureNameCode` function obtains the name code for a font's feature type or selector that matches an ASTUI font ID, feature type and feature selector values. By default, `ATSUGetFontFeatureNameCode` function obtains the name code of a feature selector. To determine the name code of a feature type, pass the constant `kATSUNoSelector` in the `iSelector` parameter.

You can use the function `ATSUFindFontName` (page 1880) to obtain the localized name string for the name code produced by `ATSUGetFontFeatureNameCode`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetFontFeatureSelectors

Obtains the available feature selectors for a given feature type in a font.

```
OSStatus ATSUGetFontFeatureSelectors (
   ATSUFontID iFontID,
   ATSUFontFeatureType iType,
   ItemCount iMaximumSelectors,
   ATSUFontFeatureSelector oSelectors[],
   Boolean oSelectorIsOnByDefault[],
   ItemCount *oActualSelectorCount,
   Boolean *oIsMutuallyExclusive
);
```

Parameters

iFont
An `ATSUFontID` value identifying the font to examine.

iType
An `ATSUFontFeatureType` value specifying one of the font's supported feature types. To obtain the available feature types for a font, call the function `ATSUGetFontFeatureTypes` (page 1898).

iMaximumSelectors
An `ItemCount` value specifying the maximum number of feature selectors to obtain for the font's specified feature type. Typically, this is equivalent to the number of elements in the `oSelectors` array.

oSelectors
A pointer to memory you have allocated for an array of `ATSUFontFeatureSelector` values. You can call the function `ATSUCountFontFeatureSelectors` (page 1859) to obtain the number of available feature selectors for a given font feature type and thus determine the amount of memory to allocate. On return, the array contains constants identifying each available feature selector for the given feature type. The constants that represent font feature selectors are defined in the header file `SFNTLayoutTypes.h` and are described in *Inside Mac OS X: Rendering Unicode Text With ATSUI*.

oSelectorIsOnByDefault
A pointer to memory you have allocated for an array of `Boolean` values. The number of elements in this array should correspond to the number of elements in the *oSelectors* array. On return, the array contains `Boolean` values indicating whether the corresponding feature selector in the *oSelectors* array is on or off. If `true`, the feature selector is on by default; if false, off.

oActualSelectorCount

A pointer to an `ItemCount` value. On return, the value specifies the actual number of feature selectors defined for the given feature type. This value may be greater than the value you specify in the `iMaximumSelectors` parameter.

oIsMutuallyExclusive

A pointer to a `Boolean` value. On return, the value indicates whether the feature selectors for the given feature type are exclusive or nonexclusive. If a feature type is exclusive you can choose only one of its available feature selectors at a time, such as whether to display numbers as proportional or fixed-width. If a feature type is nonexclusive, you can enable any number of feature selectors at once. If `true`, the feature type is exclusive and only one selector can be used at a time.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
A given font may not support all possible feature types and selectors. If you select features that are not available in a font, you won't see a change in the glyph's appearance. To determine the available features of a font, you can call the functions `ATSUGetFontFeatureTypes` (page 1898) and `ATSUGetFontFeatureSelectors`.

The `ATSUGetFontFeatureSelectors` function reads the font data table for the specified font and obtains its supported feature selectors for the given feature types. You can then use this information both to present the user a list of font features from which to select and to call such functions as `ATSUSetFontFeatures` (page 1952) with more accuracy.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetFontFeatureTypes

Obtains the available feature types of a font.

```
OSStatus ATSUGetFontFeatureTypes (
   ATSUFontID iFontID,
   ItemCount iMaximumTypes,
   ATSUFontFeatureType oTypes[],
   ItemCount *oActualTypeCount
);
```

Parameters

iFont

An `ATSUFontID` value identifying the font to examine.

iMaximumTypes

An `ItemCount` value specifying the maximum number of feature types to obtain for the font. Typically, this is equivalent to the number of elements in the `oTypes` array.

oTypes

A pointer to memory you have allocated for an array of `ATSUFontFeatureType` values. You can call the function `ATSUCountFontFeatureTypes` (page 1860) to obtain the number of available feature types for a given font and thus determine the amount of memory to allocate. On return, the array contains constants identifying each type of feature that is defined for the font. The constants that represent font feature types are defined in the header file `SFNTLayoutTypes.h` and are described in *Inside Mac OS X: Rendering Unicode Text With ATSUI*.

oActualTypeCount

A pointer to an `ItemCount` value. On return, the value specifies the actual number of feature types defined in the font. This may be greater than the value you specify in the `iMaximumTypes` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
A given font may not support all possible feature types and selectors. If you select features that are not available in a font, you won't see a change in the glyph's appearance. To determine the available features of a font, you can call the functions `ATSUGetFontFeatureTypes` and `ATSUGetFontFeatureSelectors` (page 1897).

The `ATSUGetFontFeatureTypes` function reads the font data table for the specified font and obtains its supported feature types. You can then use this information both to present the user a list of font features from which to select and to call such functions as `ATSUSetFontFeatures` (page 1952) with more accuracy.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetFontIDs

Obtains a list of all the ATSUI-compatible fonts installed on the user's system.

```
OSStatus ATSUGetFontIDs (
   ATSUFontID oFontIDs[],
   ItemCount iArraySize,
   ItemCount *oFontCount
);
```

Parameters

oFontIDs

A pointer to memory you have allocated for an array of `ATSUFontID` values. On return, the array contains unique identifiers for each of the ATSUI-compatible fonts installed on the user's system. You should allocate enough memory to contain an array the size of the count produced by the function `ATSUFontCount` (page 1885).

iArraySize

An `ItemCount` value specifying the maximum number of fonts to obtain. Typically, this is equivalent to the number of `ATSUFontID` values for which you have allocated memory in the `oFontIDs` parameter.

oFontCount

A pointer to an `ItemCount` value. On return, the value specifies the actual number of ATSUI-compatible fonts installed on the user's system. This may be greater than the value you specified in the `iArraySize` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetFontIDs` function obtains the IDs of all the fonts on the user's system except for the last-resort font. It is important to note that the set of installed ATSUI-compatible fonts may change while your application is running. In Mac OS X, the set of installed fonts may change at any time. Although in Mac OS 9, fonts cannot be removed from the Fonts folder while an application other than the Finder is running, they can be removed from other locations, and it is possible for fonts to be added.

To ensure an accurate representation of the set of installed ATSUI-compatible fonts, you should call `ATSUGetFontIDs` to rebuild your font menu each time your application is brought to the foreground.

Finally, note that Apple Type Services assigns `ATSUFontID` values systemwide at runtime. As a result, these font IDs can change across system restarts.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetFontInstance

Obtains the font variation axis values for a font instance.

```
OSStatus ATSUGetFontInstance (
   ATSUFontID iFontID,
   ItemCount iFontInstanceIndex,
   ItemCount iMaximumVariations,
   ATSUFontVariationAxis oAxes[],
   ATSUFontVariationValue oValues[],
   ItemCount *oActualVariationCount
);
```

Parameters

iFont

An `ATSUFontID` value identifying the font to examine.

iFontInstanceIndex

An `ItemCount` value specifying an index into an array of instances for the font. This index identifies the font instance to examine. Because this index is zero-based, you must pass a value between 0 and one less than the value produced in the `oInstances` parameter of the function `ATSUCountFontInstances` (page 1860).

iMaximumVariations

An ItemCount value specifying the maximum number of font variation axes to obtain for the font instance. Typically, this is equivalent to the number of ATSUFontVariationAxis and ATSUFontVariationValue values for which you have allocated memory in the oAxes and oValues parameters, respectively. To determine this value, see the Discussion.

oAxes

A pointer to memory you have allocated for an array of ATSUFontVariationAxis values. On return, the array contains tags identifying the font variation axes that constitute the font instance. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oValues

A pointer to memory you have allocated for an array of ATSUFontVariationValue values. On return, the array contains the defined values for the font variation axes produced in the oAxes array. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oActualVariationCount

A pointer to an ItemCount value. On return, the value specifies the actual number of font variation axes that constitute the font instance. This may be greater than the value you passed in the iMaximumVariations parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
A font instance consists of a named set of values for each variation axis in a font. For example, suppose a font has the variation axis 'wght' with a minimum value of 0.0, a default of 0.5, and a maximum of 1.0. Additionally, the variation axis 'wdth' is also defined for the font, with a similar value range. The type designer can then choose to declare a font instance for a set of specific values within these axes, such as "Demibold" for a value of 0.8 for the 'wght' axis and 0.5 for the 'wdth' axis. By calling the function ATSUGetFontInstance, you can obtain the variation axis values for a given index into an array of font instances.

Typically you use the function ATSUGetFontInstance by calling it twice, as follows:

1. Pass the ID of the font to examine in the iFont parameter, a valid pointer to an ItemCount value in the *oActualVariationCount* parameter, NULL for the oAxes and oValues parameters, and 0 for the other parameters. ATSUGetFontInstance returns the size to use for the oAxes and oValues arrays in the oActualVariationCount parameter.

2. Allocate enough space for arrays of the returned size, then call the ATSUGetFontInstance again, passing pointers to the arrays in the oAxes and oValues parameters. On return, the arrays contain the font variation axes and their corresponding values, respectively, for the font instance.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeFonts.h

ATSUGetFontInstanceNameCode

Obtains the name code for the font instance that matches an ASTUI font ID and font instance index value.

```
OSStatus ATSUGetFontInstanceNameCode (
   ATSUFontID iFontID,
   ItemCount iInstanceIndex,
   FontNameCode *oNameCode
);
```

Parameters

iFont

The `ATSUFontID` value of the font for which to obtain a font instance name code. Note that because Apple Type Services assigns `ATSUFontID` values systemwide at runtime, font IDs can change across system restarts.

iInstanceIndex

An `ItemCount` value providing an index to the font instance for which to obtain a name code. Because this index must be 0-based, you should pass a value between 0 and one less than the count produced by the function `ATSUCountFontInstances` (page 1860).

oNameCode

A pointer to a `FontNameCode` value. On return, the value contains the name code for the font instance. The `FontNameCode` is a `UInt32` data type, and it is defined in the `SFNTTypes.h` header file.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
A font instance consists of a named set of values for each variation axis in a font. The `ATSUGetFontInstanceNameCode` function obtains the name code for the font instance that matches an ASTUI font ID and font instance index value.

You can use the function `ATSUFindFontName` (page 1880) to obtain the localized name string for the name code produced by `ATSUGetFontInstanceNameCode`. You can obtain the font variation axis values for a font instance by calling the function`ATSUGetFontInstance` (page 1900).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetFontVariationNameCode

Obtains the name code for the font variation that matches an ASTUI font ID and font variation axis.

```
OSStatus ATSUGetFontVariationNameCode (
   ATSUFontID iFontID,
   ATSUFontVariationAxis iAxis,
   FontNameCode *oNameCode
);
```

Parameters

iFont

The `ATSUFontID` value of the font for which to obtain a font variation name code. Note that because Apple Type Services assigns `ATSUFontID` values systemwide at runtime, font IDs can change across system restarts.

iAxis

An `ATSUFontVariationAxis` value representing a valid variation axis tag. To obtain a valid variation axis tag for a font, you can call the functions`ATSUGetIndFontVariation` (page 1911) or `ATSUGetFontInstance` (page 1900).

oNameCode

A pointer to a `FontNameCode` value. On return, the value contains the name code for the font variation. The `FontNameCode` is a `UInt32` data type, and it is defined in the `SFNTTypes.h` header file.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetFontVariationNameCode` function obtains the name code for the font variation that matches an ASTUI font ID and font variation axis tag. You can use the function `ATSUFindFontName` (page 1880) to obtain the localized name string for the name code produced by `ATSUGetFontVariationNameCode`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetFontVariationValue

Obtains the current value for a font variation axis in a style object.

```
OSStatus ATSUGetFontVariationValue (
   ATSUStyle iStyle,
   ATSUFontVariationAxis iFontVariationAxis,
   ATSUFontVariationValue *oFontVariationValue
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object to examine.

iFontVariationAxis

An `ATSUFontVariationAxis` tag specifying the style object's variation axis to examine. You can obtain variation axis tags for a style object from the function `ATSUGetAllFontVariations` (page 1888).

oFontVariationValue

A pointer to memory you have allocated for an `ATSUFontVariationValue` value. On return, `ATSUGetFontVariationValue` produces the currently set value for the style object's specified variation axis. If this value has not been set, `ATSUGetFontVariationValue` produces the font-defined default value.

Return Value
A result code. Note that if no value has been set for the specified variation axis, `ATSUGetFontVariationValue` produces the font-defined default value and returns the result code `kATSUNotSetErr`. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetFontVariationValue` function obtains the setting for a specified font variation axis in a style object. You might typically call `ATSUGetFontVariationValue` if you need to obtain one previously set variation axis value after another within your program's processing loop. To obtain all nondefault font variation axis values for a given style object, you can call the function `ATSUGetAllFontVariations` (page 1888).

Before calling `ATSUGetFontVariationValue`, call the function ATSUGetAllFontVariations (page 1888) to obtain the font variation axes that are set for the style object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetGlyphBounds

Obtains the typographic bounds of a line of glyphs after final layout.

```
OSStatus ATSUGetGlyphBounds (
   ATSUTextLayout iTextLayout,
   ATSUTextMeasurement iTextBasePointX,
   ATSUTextMeasurement iTextBasePointY,
   UniCharArrayOffset iBoundsCharStart,
   UniCharCount iBoundsCharLength,
   UInt16 iTypeOfBounds,
   ItemCount iMaxNumberOfBounds,
   ATSTrapezoid oGlyphBounds[],
   ItemCount *oActualNumberOfBounds
);
```

Parameters

`iTextLayout`
An `ATSUTextLayout` value specifying the text layout object to examine.

`iTextBasePointX`
An `ATSUTextMeasurement` value specifying the x-coordinate of the origin of the line containing the glyphs in the current graphics port or Quartz graphics context. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to obtain the glyph bounds relative to the current pen location in the current graphics port or graphics context. You can pass 0 to obtain only the dimensions of the bounds relative to one another, not their actual onscreen position.

`iTextBasePointY`
An `ATSUTextMeasurement` value specifying the y-coordinate of the origin of the line containing the glyphs in the current graphics port or Quartz graphics context. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to obtain the glyph bounds relative to the current pen location in the current graphics port or graphics context. You can pass `0` to obtain only the dimensions of the bounds relative to one another, not their actual onscreen position.

iBoundsCharStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the character corresponding to the first glyph to measure. To indicate that the text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`.

iBoundsCharLength

A `UniCharCount` value specifying the length of the text range to measure. If you want the range to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

iTypeOfBounds

A glyph bounds constant indicating whether the width of the resulting typographic glyph bounds is determined using the caret origin (midway between two characters), the glyph origin in device space, or the glyph origin in fractional absolute positions (uncorrected for device display). See "Glyph Origin Selectors" (page 2049) for a description of possible values.

iMaxNumberOfBounds

An `ItemCount` value specifying the maximum number of bounding trapezoids to obtain. Typically, this is equivalent to the number of bounds in the `oGlyphBounds` array. To determine this value, see the Discussion.

oGlyphBounds

A pointer to memory you have allocated for an array of `ATSTrapezoid` values. On return, the array contains a trapezoid representing the typographic bounds for glyphs in the text range. If the specified range of text encloses nested bidirectional text, `ATSUGetGlyphBounds` produces multiple trapezoids defining these regions.In ATSUI 1.1, the maximum number of enclosing trapezoids that can be returned is 31; in ATSUI 1.2, the maximum number is 127. If you pass a range that covers an entire line, `ATSUGetGlyphBounds` returns 1 trapezoid. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oActualNumberOfBounds

A pointer to an `ItemCount` value. On return, the value specifies the actual number of enclosing trapezoids bounding the specified characters. This may be greater than the value you provide in the `iMaxNumberOfBounds` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
There are two kinds of bounds that your application may typically want to obtain for a block of text: typographic bounds and image bounds. The image bounds define the smallest rectangle that completely encloses the filled or framed parts of a block of text—that is, the text's "inked" glyphs. Because of the potential differences in glyph height in a text block, your application may instead need to determine the typographic bounds. The typographic bounding rectangle contains the extra space above and below the image bounding rectangle where characters with ascenders or descenders would be drawn (even if none currently are).

The `ATSUGetGlyphBounds` function produces the enclosing trapezoid(s) that represent the typographic bounds for glyphs in a final, laid-out range of text. You typically call this function when you need to obtain an enclosing trapezoid for a line, taking rotation and all other layout attributes into account.

ATSUI determines the height of each trapezoid by examining any line ascent and descent attribute values you may have set for the line. If you have not set these attributes for the line, the `ATSUGetGlyphBounds` function uses any line ascent and descent values you may have set for the text layout object containing the line. If these are not set, `ATSUGetGlyphBounds` uses the font's natural line ascent and descent values for the line. If these are previously set, `ATSUGetGlyphBounds` uses the `ATSUStyle` ascent and or descent/leading values.

Depending on the value you pass in the `iTypeOfBounds` parameter, the width of the resulting trapezoid(s) is determined using one of the following values:

- the caret origin, located halfway between two characters, which should be used when performing your own highlighting
- the glyph origin in device space, which is useful for obtaining bounds adjusted for specific rendering and device constraints
- the glyph origin in fractional (or "ideal") absolute positions, uncorrected for device display

Note that the coordinates produced for the trapezoid(s) are offset by the amount specified in the `iTextBasePointX` and `iTextBasePointY` parameters. If your goal in calling the `ATSUGetGlyphBounds` function is to obtain metrics for drawing the typographic bounds on the screen, pass the position of the origin of the line in the current graphics port or graphics context in these parameters. This enables `ATSUGetGlyphBounds` to match the trapezoids to their onscreen image.

Before calculating the typographic glyph bounds for the given text range, the `ATSUGetGlyphBounds` function examines the text layout object to make sure that the style runs cover the entire range of text. If there are gaps between style runs, `ATSUGetGlyphBounds` assigns the characters in the gap to the style run following the gap. If there is no style run at the beginning of the range of text, `ATSUGetGlyphBounds` assigns these characters to the first style run it can find. If there is no style run at the end of the range of text, `ATSUGetGlyphBounds` assigns the remaining characters to the last style run it can find.

Typically you use the `ATSUGetGlyphBounds` function by calling it twice, as follows:

1. Pass `NULL` for the `oGlyphBounds` parameter, `0` for the *`iMaxNumberOfBounds`* parameter, and valid values for the other parameters. The `ATSUGetGlyphBounds` function returns the actual number of trapezoids needed to enclose the glyphs in the `oActualNumberOfBounds` parameter.
2. Allocate enough space for a buffer of the returned size, then call the function again, passing a valid pointer to the buffer in the `oGlyphBounds` parameter. On return, the buffer contains the trapezoids for the glyphs' typographic bounds.

To obtain the typographic bounds of a line of text prior to line layout, call the function `ATSUGetUnjustifiedBounds` (page 1923). To calculate the image bounding rectangle for a final laid-out line, call the function `ATSUMeasureTextImage` (page 1938).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeDrawing.h`

ATSUGetGlyphInfo

Obtains a copy of the style and layout information for each glyph in a line. (Deprecated in Mac OS X v10.3. Use functions from "Accessing Glyph Data" (page 1842) instead.)

Not recommended.

```
OSStatus ATSUGetGlyphInfo (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart,
   UniCharCount iLineLength,
   ByteCount *ioBufferSize,
   ATSUGlyphInfoArray *oGlyphInfoPtr
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

iLineStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the line to examine. To indicate that the line starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iLineLength` parameter.

iLineLength

A `UniCharCount` value specifying the length of the line. If you want the line to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

ioBufferSize

A pointer to a `ByteCount` value specifying the size of the buffer you have allocated for the `ATSUGlyphInfoArray` structure produced in the *oGlyphInfoPtr* parameter. On return, the value specifies the actual size of the `ATSUGlyphInfoArray` structure.

oGlyphInfoPtr

A pointer to an `ATSUGlyphInfoArray` structure. On return, the structure contains values identifying the text layout object, the number of glyphs in the specified line, and an array of `ATSUGlyphInfo` structures for each of the glyphs. Each `ATSUGlyphInfo` structure contains information identifying the glyph, the style object with which it is associated, and other related layout values.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUGetGlyphInfo` function obtains a copy of the style and layout information for each glyph in a line of text. Copying can be slow, so it's best to use this function only if you do not plan to modify the glyph information. If you do modify the glyph information, you can only draw the modified glyphs by calling the function `ATSUDrawGlyphInfo` (page 1877). Because you are working with a copy of the glyph data and not the actual data that ATSUI has, if you try to draw text by calling the `ATSUDrawText` (page 1877) function, none of the changes you make to the glyph information will be reflected in the drawn text.

Note that is you obtain glyph information with the function `ATSUGetGlyphInfo` and then draw glyphs using `ATSUDrawGlyphInfo`, ATSUI does not take synthetic styles into account when it draw. This means that font substitution will not work.

If you want to modify glyph information you should instead use the ATSUI direct-access functions `ATSUGlyphGetQuadraticPaths` (page 1928) or `ATSUGlyphGetCubicPaths` (page 1925). You use each of these functions along with callback functions you supply for drawing the glyphs. When you modify and draw glyphs using ATSUI's direct-access functions, you obtain access to the same information as that supplied by the function `ATSUGetGlyphInfo`, but in a way that allows font substitution to work. For more information on retrieving and drawing glyph outlines, see *Inside Mac OS X: Rendering Unicode Text With ATSUI*.

The Unicode characters in the text layout object (`ATSUTextLayout`) and the glyphs returned by the function `ATSUGetGlyphInfo` do not necessarily have a one-to-one correspondence. For example, the accented Latin character é can be represented by an e with a combining ´ accent. In this case, two characters map to one glyph.

Common ligatures such as fi also form automatically for some fonts, causing two characters to map to one glyph. Right-to-left scripts such as Arabic, and complex scripts such as Devanagari or Thai have even more complicated mappings from characters to glyphs.

For this reason it's best to use the high level ATSUI functions whenever possible, and to associate a paragraph of text with a text layout object. Your application is then completely insulated from such issues.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`ATSUnicodeGlyphs.h`

ATSUGetIndFontName

Obtains a name string, name code, platform, script, and language for the font that matches an ATSUI font ID and name table index value.

```
OSStatus ATSUGetIndFontName (
   ATSUFontID iFontID,
   ItemCount iFontNameIndex,
   ByteCount iMaximumNameLength,
   Ptr oName,
   ByteCount *oActualNameLength,
   FontNameCode *oFontNameCode,
   FontPlatformCode *oFontNamePlatform,
   FontScriptCode *oFontNameScript,
   FontLanguageCode *oFontNameLanguage
);
```

Parameters

`iFontID`

The `ATSUFontID` value of the font for which to obtain information. Note that because Apple Type Services assigns `ATSUFontID` values systemwide at runtime, font IDs can change across system restarts.

`iFontNameIndex`

An `ItemCount` value providing an index to the font for which to obtain information. Because this index must be 0-based, you should pass a value between 0 and one less than the count produced by the function `ATSUCountFontNames` (page 1861).

`iMaximumNameLength`

A `ByteCount` value specifying the maximum length of the font name string to obtain. Typically, this is equivalent to the size of the buffer that you have allocated in the `oName` parameter. To determine this length, see the Discussion.

oName

A pointer to a buffer. On return, the buffer contains the name string of the font matching the ATSUI font ID and name table index value being passed. If the buffer you allocate is not large enough to contain the name string, `ATSUGetIndFontName` produces a partial string.

oActualNameLength

A pointer to a `ByteCount` value. On return, the value specifies the actual length of the complete name string. This may be greater than the value passed in the `iMaximumNameLength` parameter. You should check this value to ensure that you have allocated sufficient memory and therefore obtained the complete name string for the font.

oFontNameCode

A pointer to a `FontNameCode` value. On return, the value contains the name code for the font. The `FontNameCode` is a `UInt32` data type, and it is defined in the `SFNTTypes.h` header file. ATSUI can return any of the following constants, `kFontCopyrightName`, `kFontFamilyName`, `kFontStyleName`, `kFontUniqueName`, `kFontFullName`, `kFontVersionName`, `kFontPostscriptName`, `kFontTrademarkName`, `kFontManufacturerName`, `kFontDesignerName`, `kFontDescriptionName`, `kFontVendorURLName`, `kFontDesignerURLName`, `kFontLicenseDescriptionName`, or `kFontLicenseInfoURLName`.

oFontNamePlatform

A pointer to a `FontPlatformCode` value. On return, this value specifies the encoding of the font, for example, `kFontUnicodePlatform`, `kFontMacintoshPlatform`, `kFontReservedPlatform`, `kFontMicrosoftPlatform`, or `kFontCustomPlatform`. See the `SFNTTypes.h` header file for a definition of the `FontPlatformCode` type and a list of possible values.

oFontNameScript

A pointer to a `FontScriptCode` value. On return, this value specifies the script code of the font, for example, `kFontRomanScript`. See the `SFNTTypes.h` header file for a definition of the `FontScriptCode` type and a list of possible values.

oFontNameLanguage

A pointer to a `FontLanguageCode` value. On return, this value specifies the language of the font, for example, `kFontNorwegianLanguage`. See the `SFNTTypes.h` header file for a definition of the `FontLanguageCode` type and a list of possible values.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetIndFontName` function obtains a name string, name code, language code, script code, and platform code for the font that matches the specified ATSUI font ID and name table index value.

Typically you use the `ATSUGetIndFontName` function by calling it twice, as follows:

1. Pass valid values for the *iFontID*, *iFontNameIndex*, and *oActualNameLength* parameters, 0 for the *iMaximumNameLength* parameter, and `NULL` for the other parameters. `ATSUGetIndFontName` returns the length of the font name string in the `oActualNameLength` parameter.

2. Allocate enough space for a buffer of the returned size, then call the function again, passing a valid pointer to the buffer in the `oName` parameter. On return, the buffer contains the font name string.

To find a name string and index value for the first font in a name table that matches an ATSUI font ID and the specified font parameters, call the function `ATSUFindFontName` (page 1880). To obtain an ATSUI font ID for the first font in a name table that matches the specified name string, name code, platform, script, and/or language, call the function `ATSUFindFontFromName` (page 1879).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeFonts.h`

ATSUGetIndFontTracking

Obtains the name code and tracking value for the font tracking that matches an ASTUI font ID, glyph orientation, and tracking table index.

```
OSStatus ATSUGetIndFontTracking (
   ATSUFontID iFontID,
   ATSUVerticalCharacterType iCharacterOrientation,
   ItemCount iTrackIndex,
   Fixed *oFontTrackingValue,
   FontNameCode *oNameCode
);
```

Parameters

iFont

The `ATSUFontID` value of the font tracking for which to obtain a name code and tracking value. Note that because Apple Type Services assigns `ATSUFontID` values systemwide at runtime, font IDs can change across system restarts.

iCharacterOrientation

An `ATSUVerticalCharacterType` constant identifying the glyph orientation of the font tracking value to obtain, for example `kATSUStronglyHorizontal` or `kATSUStronglyVertical`. See "Vertical Character Types" (page 2068) for a description of possible values.

iTrackIndex

An `ItemCount` value providing an index to the font tracking for which to obtain information. Because this index must be 0-based, you should pass a value between 0 and one less than the count produced by the function `ATSUCountFontTracking` (page 1861).

oFontTrackingValue

A pointer to a `Fixed` value. On return, the value contains the font tracking value.

oNameCode

A pointer to a `FontNameCode` value. On return, the value contains the name code for the font tracking. The `FontNameCode` is a `UInt32` data type, and it is defined in the `SFNTTypes.h` header file.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You can call the `ATSUGetIndFontTracking` function to obtain the name code and tracking value that matches the specified ATSUI font ID, glyph orientation, and tracking table index value.

You can use the function `ATSUFindFontName` (page 1880) to obtain the localized name string for the name code produced by `ATSUGetIndFontTracking`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeFonts.h

ATSUGetIndFontVariation

Obtains a variation axis and its value range for a font.

```
OSStatus ATSUGetIndFontVariation (
   ATSUFontID iFontID,
   ItemCount iVariationIndex,
   ATSUFontVariationAxis *oATSUFontVariationAxis,
   ATSUFontVariationValue *oMinimumValue,
   ATSUFontVariationValue *oMaximumValue,
   ATSUFontVariationValue *oDefaultValue
);
```

Parameters

iFont

An ATSUFontID value identifying the font to examine.

iVariationIndex

An ItemCount value specifying an index into the array of variation axes for the font. This index identifies the font variation axis to examine. Because this index is zero-based, you must pass a value between 0 and one less than the value produced in the oVariationCount parameter of the function ATSUCountFontVariations (page 1862).

oATSUFontVariationAxis

A pointer to an ATSUFontVariationAxis value. On return, the value provides a four-character code identifying the font variation axis corresponding to the specified index.

oMinimumValue

A pointer to an ATSUFontVariationValue value. On return, the value identifies the variation axis minimum.

oMaximumValue

A pointer to an ATSUFontVariationValue value. On return, the value identifies the variation axis maximum.

oDefaultValue

A pointer to an ATSUFontVariationValue value. On return, the value identifies the variation axis default.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
By calling the function ATSUGetIndFontVariation, you can obtain a variation axis and its maximum, minimum, and default values for a font.

If you supply font variation axes and values to the function ATSUSetVariations (page 1967), you can change the appearance of a style object's font accordingly.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeFonts.h

ATSUGetLayoutControl

Obtains a layout control attribute value for a text layout object.

```
OSStatus ATSUGetLayoutControl (
   ATSUTextLayout iTextLayout,
   ATSUAttributeTag iTag,
   ByteCount iExpectedValueSize,
   ATSUAttributeValuePtr oValue,
   ByteCount *oActualValueSize
);
```

Parameters

iTextLayout

An ATSUTextLayout value specifying the text layout object for which to obtain a layout control attribute value.

iTag

An ATSUAttributeTag constant identifying the attribute value to obtain. See "Attribute Tags" (page 2030) for a description of the Apple-defined attribute tag constants.

iExpectedValueSize

The expected size (in bytes) of the value to obtain. To determine the size of an application-defined style attribute value, see the Discussion.

oValue

An ATSUAttributeValuePtr pointer, identifying the memory you have allocated for the attribute value. If you are uncertain of how much memory to allocate, see the Discussion. On return, *oValue* contains a valid pointer to the actual attribute value. If the value is unset, ATSUGetLayoutControl produces the default value in this parameter.

oActualValueSize

A pointer to a ByteCount value. On return, the value contains the actual size (in bytes) of the attribute value. You should examine this parameter if you are unsure of the size of the attribute value being obtained.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The ATSUGetLayoutControl function obtains the value of a specified layout control attribute for a given text layout object.

Before calling ATSUGetLayoutControl, you should call the function ATSUGetAllLayoutControls (page 1890) to obtain an array of nondefault layout control attribute tags and value sizes for the text layout object. You can then pass the tag and value size for the attribute value to obtain to ATSUGetLayoutControl.

Typically you use the function ATSUGetLayoutControl by calling it twice, as follows:

1. Pass a reference to the text layout object to examine in the iTextLayout parameter, NULL for the oValue parameter, 0 for the iExpectedValueSize parameter. ATSUGetLayoutControl returns the actual size of the attribute value in the oActualValueSize parameter.

2. Allocate enough space for an array of the returned size, then call the `ATSUGetLayoutControl` function again, passing a valid pointer in the `oValue` parameter. On return, the pointer refers to the actual attribute value contained in the text layout object.

To obtain the value of a line control attribute value for a text layout object, call the function `ATSUGetLineControl` (page 1913).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetLineControl

Obtains a line control attribute value for a line in a text layout object.

```
OSStatus ATSUGetLineControl (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart,
   ATSUAttributeTag iTag,
   ByteCount iExpectedValueSize,
   ATSUAttributeValuePtr oValue,
   ByteCount *oActualValueSize
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object for which to obtain a line control attribute value.

iLineStart
A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the line for which to obtain a line control attribute value.

iTag
An `ATSUAttributeTag` constant identifying the attribute value to obtain. See "Attribute Tags" (page 2030) for a description of the Apple-defined attribute tag constants.

iExpectedValueSize
The expected size (in bytes) of the value to obtain.

oValue
An `ATSUAttributeValuePtr` pointer, identifying the memory you have allocated for the attribute value. If you are uncertain of how much memory to allocate, see the Discussion. On return, *oValue* contains a valid pointer to the actual attribute value. If the value is unset, `ATSUGetLineControl` produces the default value in this parameter.

oActualValueSize
A pointer to a `ByteCount` value. On return, the value contains the actual size (in bytes) of the attribute value. You should examine this parameter if you are unsure of the size of the attribute value being obtained.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetLineControl` function obtains the value of a specified line control attribute for a given line of text in a text layout object.

Before calling `ATSUGetLineControl`, you should call the function `ATSUGetAllLineControls` (page 1891) to obtain an array of nondefault line control attribute tags and value sizes for the line. You can then pass the tag and value size for the attribute value to obtain to `ATSUGetLineControl`.

To obtain the value of a layout control attribute value for a text layout object, call the function `ATSUGetLayoutControl` (page 1912).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetNativeCurveType

Obtains the type of outline path used for glyphs associated with a given style object.

```
OSStatus ATSUGetNativeCurveType (
   ATSUStyle iATSUStyle,
   ATSCurveType *oCurveType
);
```

Parameters

iATSUStyle
An `ATSUStyle` value specifying the style object to examine.

oCurveType
A pointer to an `ATSCurveType` value. On return, the value provides a constant specifying the type of outline path being used. Possible values include `kATSCubicCurveType`, `kATSQuadCurveType`, and `kATSOtherCurveType`.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You can call the `ATSUGetNativeCurveType` function to obtain the type of outline path used for glyphs associated with a given style object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeGlyphs.h`

ATSUGetObjFontFallbacks

Obtains the font list and font-search method associated with a font fallback object.

```
OSStatus ATSUGetObjFontFallbacks (
   ATSUFontFallbacks iFontFallbacks,
   ItemCount iMaxFontFallbacksCount,
   ATSUFontID oFonts[],
   ATSUFontFallbackMethod *oFontFallbackMethod,
   ItemCount *oActualFallbacksCount
);
```

Parameters

iFontFallbacks

An `ATSUFontFallbacks` value specifying the font fallback object to examine.

iMaxFontFallbacksCount

An `ItemCount` value specifying the maximum number of fonts that you want to obtain. Typically, this is equivalent to the size of the array allocated in the `oFonts` parameter. To determine this value, see the Discussion.

oFonts

A pointer to memory you have allocated for an array of `ATSUFontID` values. If you are uncertain of how much memory to allocate, see the Discussion. On return, the array contains font IDs identifying the fonts in the font list associated with the font fallback object.

oFontFallbackMethod

A pointer to an `ATSUFontFallbackMethod` value. On return, the value identifies the font-search method associated with the font fallback object. See "Font Fallback Methods" (page 2048) for a description of possible values.

oActualFallbacksCount

A pointer to an `ItemCount` value. On return, the value specifies the actual number of fonts in the font list associated with the text layout object. This value may be greater than that passed in the `iMaxFontFallbacksCount` parameter.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUGetObjFontFallbacks` function obtains the list of fonts and the search order associated with a given font fallback object.

Typically you use the function `ATSUGetObjFontFallbacks` by calling it twice, as follows:

1. Pass valid values for the *iFontFallbacks* and *oActualFallbacksCount* parameters, `NULL` for the `oFonts` and *oFontFallbackMethod* parameters and `0` for the *iMaxFontFallbacksCount* parameter. `ATSUGetObjFontFallbacks` returns the size of the font array in the `oActualFallbacksCount` parameter.

2. Allocate enough space for an array of the returned size, then call the function again, passing a valid pointer in the `oFonts` parameter. On return, the array contains the font list associated with the font fallback object.

You set the font list and search method for a font fallback object by calling the function `ATSUSetObjFontFallbacks` (page 1958).

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In
ATSUnicodeObjects.h

ATSUGetRunStyle

Obtains style run information for a character offset in a run of text.

```
OSStatus ATSUGetRunStyle (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOffset,
   ATSUStyle *oStyle,
   UniCharArrayOffset *oRunStart,
   UniCharCount *oRunLength
);
```

Parameters

iTextLayout

An ATSUTextLayout value specifying the text layout object for which to obtain style run information.

iOffset

A pointer to a UniCharArrayOffset value. This value should specify the offset from the beginning of the text buffer to the character for which to obtain style run information. To specify the beginning of the text buffer, you can pass the constant kATSUFromTextBeginning.

oStyle

A pointer to an ATSUStyle value. On return, the value specifies the style object assigned to the range of text containing the character at *iOffset*. Note that if you pass an offset in the iOffset parameter that is at a style run boundary, ATSUGetRunStyle produces style run information for the following, not preceding, style run.

oRunStart

A pointer to a UniCharArrayOffset value. On return, the value specifies the offset from the beginning of the text buffer to the first character of the style run containing the character at iOffset. Note that the entire style run does not necessarily share the same unset attribute values as the character at iOffset.

oRunLength

A pointer to a UniCharCount value. On return, the value specifies the length of the style run containing the character at iOffset.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You can use the ATSUGetRunStyle function to obtain the style object assigned to a given text offset. ATSUGetRunStyle also produces the encompassing text range that shares the style object with the offset.

Note that the style object contains those previously set style attributes, font features, and font variations that are continuous for the range of text that includes the specified text offset. If you want to obtain all shared style information for a style run, including any unset attributes, call the function ATSUGetContinuousAttributes (page 1893) instead.

If only one style run is set in the text layout object, and it does not cover the entire text layout object, ATSUGetRunStyle uses the style run information for the iOffset parameter to set the style run information for the remaining text.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetSoftLineBreaks

Obtains soft line breaks in a range of text.

```
OSStatus ATSUGetSoftLineBreaks (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iRangeStart,
   UniCharCount iRangeLength,
   ItemCount iMaximumBreaks,
   UniCharArrayOffset oBreaks[],
   ItemCount *oBreakCount
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

iRangeStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the text range to examine. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`, To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iRangeLength` parameter.

iRangeLength

A `UniCharCount` value specifying the length of the text range. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

iMaximumBreaks

An `ItemCount` value specifying the maximum number of soft line breaks to obtain. Typically, this is equivalent to the number of `UniCharArrayOffset` values for which you have allocated memory in the `oBreaks` array. To determine this value, see the Discussion.

oBreaks

A pointer to memory you have allocated for an array of `UniCharArrayOffset` values. On return, the array contains offsets from the beginning of the text buffer to each of the soft line breaks in the text range. If you are uncertain of how much memory to allocate for this array, see the Discussion.

oBreakCount

A pointer to an `ItemCount` value. On return, the value specifies the actual number of soft line breaks in the range of text. This may be greater than the value you specified in the `iMaximumBreaks` parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetSoftLineBreaks` function obtains the soft line breaks that are currently set in a given text range.

Typically you use the function `ATSUGetSoftLineBreaks` by calling it twice, as follows:

1. Pass valid values for the *`iTextLayout`*, *`iRangeStart`*, *`iRangeLength`*, and *`oBreakCount`* parameters. Pass `NULL` for the `oBreaks` parameter and 0 for the *`iMaximumBreaks`* parameter. On return, the value of the `oBreakCount` parameter specifies the number of items in the offset array.

2. Allocate enough space for an array of the appropriate size (number of items in the array multiplied by 4 bytes per item), then call the function again, passing a valid pointer in the `oBreaks` parameter. On return, the pointer refers to an array containing the text range's soft line breaks.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUGetStyleRefCon

Obtains application-specific data for a style object.

```
OSStatus ATSUGetStyleRefCon (
   ATSUStyle iStyle,
   URefCon *oRefCon
);
```

Parameters

`iStyle`
An `ATSUStyle` value specifying the style object for which to obtain application-specific data.

`oRefCon`
A pointer to a 32-bit value. On return, the value contains or refers to application-specific style data.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetStyleRefCon` function obtains a reference constant (that is, application-specific data) associated with a style object. To associate a reference constant with a style object, call the function `ATSUSetStyleRefCon` (page 1961).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetTabArray

Retrieves the tab ruler associated with a text layout object.

```
OSStatus ATSUGetTabArray (
   ATSUTextLayout iTextLayout,
   ItemCount iMaxTabCount,
   ATSUTab oTabs[],
   ItemCount *oTabCount
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object whose tab ruler you want to obtain.

iMaxTabCount

The maximum number of tabs that can be written to the `iTabs` array.

oTabs[]

An array of `ATSUTab` values. On return, this array contains the current tab values in order of position along the line from left to right. Pass `NULL` if you want to retrieve the number of tabs, but not the tab values.

oTabCount

The number of tabs set for the text layout object.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

This function can be used to retrieve all the tabs that were previously set for a text layout object, using the function `ATSUSetTabArray`. All the returned tabs will be in order of position along the line.Typically you use the `ATSUGetTabArray` function by calling it twice, as follows:

1. Pass `NULL` for the `oTabs` parameter, `0` for the *iMaxTabCount* parameter, and valid values for the other parameters. The `ATSUGetTabArray` function returns the actual number of tabs in the `oTabCount` parameter.

2. Allocate enough space for a buffer of the returned size, then call the function again, passing a valid pointer to the buffer in the `oTabs` parameter. On return, the buffer contains the tab values in order of position along the line from left to right.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUGetTextHighlight

Obtains the highlight region for a range of text.

```
OSStatus ATSUGetTextHighlight (
   ATSUTextLayout iTextLayout,
   ATSUTextMeasurement iTextBasePointX,
   ATSUTextMeasurement iTextBasePointY,
   UniCharArrayOffset iHighlightStart,
   UniCharCount iHighlightLength,
   RgnHandle oHighlightRegion
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object containing the text range.

iTextBasePointX

An `ATSUTextMeasurement` value specifying the x-coordinate of the origin (in either the current graphics port or in a Quartz graphics context) of the line containing the text range. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to obtain the highlight region relative to the current pen location in the current graphics port.

iTextBasePointY

An `ATSUTextMeasurement` value specifying the y-coordinate of the origin (in either the current graphics port or graphics context) of the line containing the text range. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to obtain the highlight region relative to the current pen location in the current graphics port.

iHighlightStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the range. If the range of text spans multiple lines, you should call `ATSUGetTextHighlight` for each line, passing the offset corresponding to the beginning of the new line with each call. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iHighlightLength` parameter.

iHighlightLength

A `UniCharCount` value specifying the length of the text range. If you want the text range to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

oHighlightRegion

A valid `RgnHandle` value. On return, `ATSUGetTextHighlight` produces a `MacRegion` structure containing the highlight region for the specified range of text. In the case of discontinuous highlighting, the region consists of multiple components, with `MacRegion.rgnBBox` specifying the bounding box around the entire area of discontinuous highlighting.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUGetTextHighlight` function obtains the highlight region for a range of text. To highlight text, call the function `ATSUHighlightText` (page 1931).

The `ATSUGetTextHighlight` function uses the previously set line ascent and descent values to calculate the height of the highlight region. If these values have not been set for the line, `ATSUGetTextHighlight` uses the line ascent and descent values set for the text layout object containing the line. If these are not set, it uses the default values.

Version Notes
When there are discontinuous highlighting regions, the structure produced in the `oHighlightRegion` parameter is made up of multiple components. In ATSUI 1.1, the maximum number of components that can be produced is 31. In ATSUI 1.2, the maximum number of components is 127.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUGetTextLayoutRefCon

Obtains application-specific data for a text layout object.

```
OSStatus ATSUGetTextLayoutRefCon (
   ATSUTextLayout iTextLayout,
   URefCon *oRefCon
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object for which to obtain application-specific data.

oRefCon
A pointer to a 32-bit value. On return, the value contains or refers to application-specific text layout data.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUGetTextLayoutRefCon` function obtains a reference constant (that is, application-specific data) associated with a text layout object. To associate a reference constant with a text layout object, call the function `ATSUSetTextLayoutRefCon` (page 1964).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUGetTextLocation

Obtains information about the text associated with a text layout object.

```
OSStatus ATSUGetTextLocation (
   ATSUTextLayout iTextLayout,
   void **oText,
   Boolean *oTextIsStoredInHandle,
   UniCharArrayOffset *oOffset,
   UniCharCount *oTextLength,
   UniCharCount *oTextTotalLength
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

oText

A pointer to data of any type. On return, the pointer is set to either a pointer or a handle that refers to the text buffer for the specified text layout object.

oTextIsStoredInHandle

A pointer to a `Boolean` value. On return, the value is set to `true` if the text buffer in the `oText` parameter is accessed by a handle; if `false`, a pointer.

oOffset

A pointer to a `UniCharArrayOffset` value. On return, the value specifies the offset from the beginning of the text buffer to the first character of the layout's current text range.

oTextLength

A pointer to a `UniCharCount` value. On return, the value specifies the length of the text range.

oTextTotalLength

A pointer to a `UniCharCount` value. On return, the value specifies the length of the entire text buffer.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

When you call the `ATSUGetTextLocation` function for a given text layout object, ATSUI obtains the location of the text layout object's associated text in physical memory, the length of the text range and its text buffer, and whether the text is accessed by a pointer or handle.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUGetTransientFontMatching

Obtains whether ATSUI automatically performs font substitution for a text layout object.

```
OSStatus ATSUGetTransientFontMatching (
   ATSUTextLayout iTextLayout,
   Boolean *oTransientFontMatching
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

oTransientFontMatching

A pointer to a `Boolean` value. On return, the value indicates whether ATSUI performs automatic font substitution for the text layout object. If `true`, ATSUI automatically performs font substitution for the text range associated with the text layout object.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

You can call the `ATSUGetTransientFontMatching` function to find out whether ATSUI automatically performs font substitution for a given text layout object when a character cannot be drawn with the assigned font. To turn automatic font substitution on or off for a text layout object, call the function `ATSUSetTransientFontMatching` (page 1967).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUGetUnjustifiedBounds

Obtains the typographic bounding rectangle for a line of text prior to final layout.

```
OSStatus ATSUGetUnjustifiedBounds (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart,
   UniCharCount iLineLength,
   ATSUTextMeasurement *oTextBefore,
   ATSUTextMeasurement *oTextAfter,
   ATSUTextMeasurement *oAscent,
   ATSUTextMeasurement *oDescent
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

iLineStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the line. To indicate that the line starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iLineLength` parameter.

iLineLength

A `UniCharCount` value specifying the length of the line. If you want the line to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

oTextBefore

A pointer to an `ATSUTextMeasurement` value. On return, the value specifies the starting point of the typographic bounds for the line, relative to the origin (0,0) of the line and taking into account cross-stream shifting. Note that the `ATSUMeasureText` function might produce negative values for the typographic starting point of the line if, for example, the initial character of the line is allowed to hang into the margin. For horizontal text, this value corresponds to the left side of the bounding rectangle.

oTextAfter

A pointer to an `ATSUTextMeasurement` value. On return, the value specifies the end point of the typographic bounds for the line, relative to the origin (0,0) of the line and taking into account cross-stream shifting. For horizontal text, this value corresponds to the right side of the bounding rectangle.

oAscent

A pointer to an `ATSUTextMeasurement` value. On return, the value specifies the ascent of the typographic bounds for the line, relative to the origin (0,0) of the line and taking into account cross-stream shifting. For horizontal text, this value corresponds to the top side of the bounding rectangle.

oDescent

A pointer to an `ATSUTextMeasurement` value. On return, the value specifies the descent of the typographic bounds for the line, relative to the origin (0,0) of the line and taking into account cross-stream shifting. For horizontal text, this value corresponds to the bottom side of the bounding rectangle.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

There are two kinds of bounds that your application may typically want to obtain for a block of text: typographic bounds and image bounds. The image bounds define the smallest rectangle that completely encloses the filled or framed parts of a block of text—that is, the text's "inked" glyphs. Because of the potential differences in glyph height in a text block, your application may instead need to determine the typographic bounds. The typographic bounding rectangle contains the extra space above and below the image bounding rectangle where characters with ascenders or descenders would be drawn (even if none currently are).

The `ATSUGetUnjustifiedBounds` function calculates the typographic bounds (in coordinates independent of the rendering device) for a line of text. Note that `ATSUGetUnjustifiedBounds` calculates these bounds prior to the text's final layout, and therefore, the calculated bounds might not reflect those of the final laid-out line. To obtain the typographic bounds of a line after it is laid out, you can call the function `ATSUGetGlyphBounds` (page 1904).

The `ATSUGetUnjustifiedBounds` function ignores any previously set line attributes such as line rotation, flushness, justification, ascent, and descent in its calculations. You typically only call `ATSUGetUnjustifiedBounds` when you need to find out what the width of a line is without these attributes, such as for determining your own line breaks or the leading and line spacing to impose on a line.

The `ATSUGetUnjustifiedBounds` function treats the specified text range as a single line. That is, if the range of text you specify is less than a line, it nevertheless treats the initial character in the range as the start of a line, for measuring purposes. If the range of text extends beyond a line, `ATSUGetUnjustifiedBounds` ignores soft line breaks, again, treating the text as a single line.

Before calculating the typographic bounds for the text range, the `ATSUGetUnjustifiedBounds` function examines the text layout object to ensure that each of the characters in the range is assigned to a style run. If there are gaps between style runs, `ATSUGetUnjustifiedBounds` assigns the characters in the gap to the style run that precedes (in storage order) the gap. If there is no style run at the beginning of the text range, `ATSUGetUnjustifiedBounds` assigns these characters to the first style run it finds. If there is no style run at the end of the text range, `ATSUGetUnjustifiedBounds` assigns the remaining characters to the last style run it finds.

To obtain the image bounding rectangle of a laid-out line, call the function `ATSUMeasureTextImage` (page 1938).

Version Notes
As of ASTUI version 2.4, this function replaces the `ATSUMeasureText` function.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUGlyphGetCubicPaths

Obtains the cubic outline paths for a glyph.

```
OSStatus ATSUGlyphGetCubicPaths (
   ATSUStyle iATSUStyle,
   GlyphID iGlyphID,
   ATSCubicMoveToUPP iMoveToProc,
   ATSCubicLineToUPP iLineToProc,
   ATSCubicCurveToUPP iCurveToProc,
   ATSCubicClosePathUPP iClosePathProc,
   void *iCallbackDataPtr,
   OSStatus *oCallbackResult
);
```

Parameters

iATSUStyle
An `ATSUStyle` value specifying the style object to examine.

iGlyphID
A `GlyphID` value identifying the glyph for which to obtain an outline path.

iMoveToProc
A pointer to your callback function for handling the pen move-to operation.

iLineToProc
A pointer to your callback function for handling the line-to operation.

iCurveToProc
A pointer to your callback function for handling the curve-to operation.

iClosePathProc
A pointer to your callback function for handling the close-path operation.

iCallbackDataPtr

A pointer to any data your callback functions need. This pointer is passed through to your callback functions.

oCallbackResult

On output, a value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetCubicPaths` function stops parsing the glyph path outline and returns the result `kATSOutlineParseAbortedErr`.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The glyph outlines that are returned are the hinted outlines at the font size specified in the style object. If you want to use unhinted outlines, set the font size to a very large size, (for example, 1000 points) and then scale down the returned curves to the desired size.

As of Mac OS X version 10.1, the curves returned by this function are derived from quadratic curves, irrespective of the native curve type of the font.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeGlyphs.h`

ATSUGlyphGetCurvePaths

Obtains the outline paths for a glyph associated with a given style object.

```
OSStatus ATSUGlyphGetCurvePaths (
   ATSUStyle iATSUStyle,
   GlyphID iGlyphID,
   ByteCount *ioBufferSize,
   ATSUCurvePaths *oPaths
);
```

Parameters

iATSUStyle

An `ATSUStyle` value specifying the style object to examine.

iGlyphID

A `GlyphID` value identifying the glyph for which to obtain an outline path.

ioBufferSize

A pointer to a `ByteCount` value specifying the size of the buffer you have allocated for the `ATSUCurvePaths` structure in the *oPaths* parameter. On return, the value provides the actual size of buffer needed to contain the produced `ATSUCurvePaths` structure.

oPaths

A pointer to an `ATSUCurvePaths` structure. On return, the `ATSUCurvePaths` structure contains a value specifying the number of contours that comprise the glyph's outline, as well as an array of `ATSUCurvePath` structures, each of which defines a contour.

Return Value
A result code. See "ATSUI Result Codes" (page 2068). If the font is a protected font, returns `kATSUInvalidFontErr`.

Discussion
This function only returns quadratic paths. The glyph outlines that are returned are the hinted outlines at the font size specified in the style object. If you want to obtain unhinted outlines, set the font size to a very large size, (for example, 1000 points) and then scale down the returned curves to the desired size. More typically, however, you would use the functions `ATSUGlyphGetCubicPaths` (page 1925) and `ATSUGlyphGetQuadraticPaths` (page 1928) when drawing curves.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeGlyphs.h`

ATSUGlyphGetIdealMetrics

Obtains resolution-independent font metric information for glyphs associated with a given style object.

```
OSStatus ATSUGlyphGetIdealMetrics (
   ATSUStyle iATSUStyle,
   ItemCount iNumOfGlyphs,
   GlyphID iGlyphIDs[],
   ByteOffset iInputOffset,
   ATSGlyphIdealMetrics oIdealMetrics[]
);
```

Parameters

iATSUStyle
An `ATSUStyle` value specifying the style object to examine.

iNumOfGlyphs
An `ItemCount` value specifying the number of glyphs to examine. This value should be the same as the number of glyph IDs being passed in the *iGlyphIDs* parameter and the number of `ATSGlyphIdealMetrics` structures for which memory is allocated in the *oIdealMetrics* parameter.

iGlyphIDs
A pointer to the first `GlyphID` value in an array of glyph IDs. Each ID should identify a glyph for which to obtain font metric information.

iInputOffset
A `ByteOffset` value specifying the offset in bytes between glyph IDs in the *iGlyphIDs* array.

oIdealMetrics
A pointer to memory you have allocated for an array of `ATSGlyphIdealMetrics` structures. On return, each structure contains advance and side-bearing values for a glyph.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The advance width is the full horizontal width of the glyph as measured from its origin to the origin of the next glyph on the line, including the left-side and right-side bearings. For vertical text, the advance height is the sum of the top-side bearing, the bounding-box height, and the bottom-side bearing.

You can call the `ATSUGlyphGetIdealMetrics` function to obtain an array of `ATSGlyphIdealMetrics` structures containing values for the specified glyphs' advance and side bearings. `ATSUGlyphGetIdealMetrics` can analyze both horizontal and vertical text, automatically producing the appropriate bearing values (oriented for width or height, respectively) for each.

You should call `ATSUGlyphGetIdealMetrics` to obtain resolution-independent glyph metrics. To obtain device-adjusted (that is, resolution-dependent) glyph metrics, call the function `ATSUGlyphGetScreenMetrics` (page 1929).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeGlyphs.h`

ATSUGlyphGetQuadraticPaths

Obtains the quadratic outline paths for a glyph.

```
OSStatus ATSUGlyphGetQuadraticPaths (
   ATSUStyle iATSUStyle,
   GlyphID iGlyphID,
   ATSQuadraticNewPathUPP iNewPathProc,
   ATSQuadraticLineUPP iLineProc,
   ATSQuadraticCurveUPP iCurveProc,
   ATSQuadraticClosePathUPP iClosePathProc,
   void *iCallbackDataPtr,
   OSStatus *oCallbackResult
);
```

Parameters

iATSUStyle
An `ATSUStyle` value specifying the style object to examine.

iGlyphID
A `GlyphID` value identifying the glyph for which to obtain an outline path.

iNewPathProc
A pointer to your callback function for handling the new-path operation.

iLineProc
A pointer to your callback function for handling the line operation.

iCurveProc
A pointer to your callback function for handling the curve operation.

iClosePathProc
A pointer to your callback function for handling the close-path operation.

iCallbackDataPtr
A pointer to any data your callback functions need. This pointer is passed through to your callback functions.

oCallbackResult
On output, a value that indicates the status of your callback function. When a callback function returns any value other than 0, the ATSGlyphGetQuadraticPaths function stops parsing the path outline and returns the result kATSOutlineParseAbortedErr.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The glyph outlines that are returned are the hinted outlines at the font size specified in the style object. If you want to use unhinted outlines, set the font size to a very large size, (for example, 1000 points) and then scale down the returned curves to the desired size.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeGlyphs.h

ATSUGlyphGetScreenMetrics

Obtains device-adjusted font metric information for glyphs associated with a given style object.

```
OSStatus ATSUGlyphGetScreenMetrics (
   ATSUStyle iATSUStyle,
   ItemCount iNumOfGlyphs,
   GlyphID iGlyphIDs[],
   ByteOffset iInputOffset,
   Boolean iForcingAntiAlias,
   Boolean iAntiAliasSwitch,
   ATSGlyphScreenMetrics oScreenMetrics[]
);
```

Parameters

iATSUStyle
An ATSUStyle value specifying the style object to examine.

iNumOfGlyphs
An ItemCount value specifying the number of glyphs to examine. This value should be the same as the number of glyph IDs being passed in the *iGlyphIDs* parameter and the number of ATSGlyphScreenMetrics structures for which memory is allocated in the *oScreenMetrics* parameter.

iGlyphIDs
A pointer to the first GlyphID value in an array of glyph IDs. Each ID should identify a glyph for which to obtain font metric information.

iInputOffset
A ByteOffset value specifying the offset in bytes between glyph IDs in the *iGlyphIDs* array.

iForcingAntiAlias
A Boolean value indicating whether anti-aliasing is forced for the style object.

iAntiAliasSwitch
A Boolean value indicating whether anti-aliasing is currently on or off.

oScreenMetrics

A pointer to memory you have allocated for an array of `ATSGlyphScreenMetrics` structures. On return, each structure contains device-adjusted metrics for a glyph, including advance and side bearings, but also values for the top left, height, and width of the glyph.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You can call the `ATSUGlyphGetScreenMetrics` function to obtain an array of `ATSGlyphScreenMetrics` structures containing values for the specified glyphs' advance and side bearings, top left, height, and width.

You should call `ATSUGlyphGetScreenMetrics` to obtain device-adjusted (that is, resolution-dependent) glyph metrics. To obtain resolution-independent glyph metrics, call the function `ATSUGlyphGetIdealMetrics` (page 1927).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeGlyphs.h`

ATSUHighlightInactiveText

Highlights previously selected text using an alpha value of 0.5.

```
OSStatus ATSUHighlightInactiveText (
   ATSUTextLayout iTextLayout,
   ATSUTextMeasurement iTextBasePointX,
   ATSUTextMeasurement iTextBasePointY,
   UniCharArrayOffset iHighlightStart,
   UniCharCount iHighlightLength
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object containing the text range.

iTextBasePointX

An `ATSUTextMeasurement` value specifying the x-coordinate of the origin (in either the current graphics port or in a Quartz graphics context) of the line containing the text range. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to obtain the highlight region relative to the current pen location in the current graphics port.

iTextBasePointY

An `ATSUTextMeasurement` value specifying the y-coordinate of the origin (in either the current graphics port or graphics context) of the line containing the text range. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to obtain the highlight region relative to the current pen location in the current graphics port.

iHighlightStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the range. If the range of text spans multiple lines, you should call `ATSUGetTextHighlight` for each line, passing the offset corresponding to the beginning of the new line with each call. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iHighlightLength` parameter.

iHighlightLength

A `UniCharCount` value specifying the length of the text range. If you want the text range to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUHighlightText

Renders a highlighted range of text at a specified location in a QuickDraw graphics port or Quartz graphics context.

```
OSStatus ATSUHighlightText (
   ATSUTextLayout iTextLayout,
   ATSUTextMeasurement iTextBasePointX,
   ATSUTextMeasurement iTextBasePointY,
   UniCharArrayOffset iHighlightStart,
   UniCharCount iHighlightLength
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object for which to render highlighted text.

iTextBasePointX

An `ATSUTextMeasurement` value specifying the x-coordinate of the origin (in either the current graphics port or in a Quartz graphics context) of the line containing the text range to highlight. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to draw relative to the current pen location in the current graphics port.

iTextBasePointY

An `ATSUTextMeasurement` value specifying the y-coordinate of the origin (in either the current graphics port or graphics context) of the line containing the text range to highlight. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), to draw relative to the current pen location in the current graphics port.

iHighlightStart

A UniCharArrayOffset value specifying the offset from the beginning of the text buffer to the first character of the range to highlight. If the range of text spans multiple lines, you should call ATSUHighlightText for each line, passing the offset corresponding to the beginning of the new line to draw with each call. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant kATSUFromTextBeginning. To specify the entire text buffer, pass kATSUFromTextBeginning in this parameter and kATSUToTextEnd in the iHighlightLength parameter.

iHighlightLength

A UniCharCount value specifying the length of the text range to highlight. If you want the range of text to extend to the end of the text buffer, you can pass the constant kATSUToTextEnd.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
When the user selects a series of glyphs, the characters in memory corresponding to the glyphs make up the selection range and should be highlighted to indicate where the next editing operation is to occur. The characters in a selection range are always contiguous in memory, but their corresponding glyphs are not necessarily so onscreen. If the selection range crosses a direction boundary, it is appropriate to display discontinuous highlighting.

The ATSUHighlightText function renders a highlighted range of text at a specified location in a QuickDraw graphics port or Quartz graphics context, using the highlight information in the graphics port or context. ATSUHighlightText automatically produces discontinuous highlighting, if needed. You typically call the ATSUHighlightText function every time you need to draw or redraw highlighted text.

If you provide your own CGContextRef (for example, one created by calling the function QDBeginCGContext) for an ATSUTextLayout, highlighting performed by calling the function ATSUHighlightText will not work unless you first call the function ATSUSetHighlightingMethod with the *iMethod* parameter set to kRedrawHighlighting and a pointer to an ATSUUnhighlightData structure as the iUnhighlightData parameter.

Before drawing the highlighted text, ATSUHighlightText examines the text layout object to ensure that each of the characters in the range is assigned to a style run. If there are gaps between style runs, ATSUI assigns the characters in the gap to the style run that precedes (in storage order) the gap. If there is no style run at the beginning of the text range, ATSUI assigns these characters to the first style run it finds. If there is no style run at the end of the text range, ATSUI assigns the remaining characters to the last style run it finds.

ATSUHighlightText uses the previously set line ascent and descent values to calculate the height of the highlighted region. If these values have not been set for the line, ATSUHighlightText uses the line ascent and descent values set for the text layout object containing the line. If these are not set, it uses the default values.

To draw a highlighted text range that spans multiple lines, you should call ATSUHighlightText for each line of the text range, even if all the lines are in the same text layout object. You should adjust the iHighlightStart parameter to reflect the beginning of each line to be drawn.

After calling ATSUHighlightText, to properly redraw the unhighlighted text and background, you should always call the function ATSUUnhighlightText (page 1974).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeDrawing.h

ATSUIdle

Performs background processing. (Deprecated in Mac OS X v10.0. There is no replacement because this function does nothing in Mac OS X.)

Not recommended.

```
OSStatus ATSUIdle (
   ATSUTextLayout iTextLayout
);
```

Parameters

iTextLayout

A reference to the text layout object in which you want ATSUI to perform background processing.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The function ATSUIdle is not recommended. Current versions of ATSUI do not implement background processing for text layout objects. In Mac OS X, the function ATSUIdle does nothing.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.0.
Not available to 64-bit applications.

Declared In
ATSUnicodeObjects.h

ATSULeftwardCursorPosition

Obtains the memory offset for the insertion point to the left of the high caret position, as determined by a move of the specified length at a line direction boundary.

```
OSStatus ATSULeftwardCursorPosition (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOldOffset,
   ATSUCursorMovementType iMovementType,
   UniCharArrayOffset *oNewOffset
);
```

Parameters

iTextLayout

An ATSUTextLayout value identifying the text layout object to examine.

iOldOffset

A `UniCharArrayOffset` value specifying the memory offset corresponding to the current caret position. To specify the beginning of the text buffer, pass the constant `kATSUFromTextBeginning`. For bidirectional text, you can specify the previous layout by passing the constant `kATSUFromPreviousLayout` and the following layout by passing the constant `kATSUFromFollowingLayout`. See the Discussion for example code that shows how to use these constants.

iMovementType

An `ATSUCursorMovementType` constant identifying the unit of movement. See "Caret Movement Types" (page 2042) for a description of possible values (which range from a single Unicode character to a Unicode word in length). Note that ATSUI may not be able to move the caret by a single Unicode character in some cases, since doing so might place the insertion point in the middle of a surrogate pair.

oNewOffset

A pointer to a `UniCharArrayOffset` value. On return, the value provides the memory offset corresponding to the new insertion point. This offset may be outside the initial text buffer.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
Line direction boundaries can occur on the trailing edges of two glyphs, the leading edges of two glyphs, or at the beginning or end of a text segment. At direction boundaries, a single insertion point in memory can require two caret positions onscreen, one for text entry in each direction. The two separate carets (known as a split caret or a dual caret) consist of a high caret and a low caret. The high (primary) caret is displayed at the caret position for inserting text whose direction corresponds to the line direction (the dominant direction for the overall line of text). The low (secondary) caret is displayed at the caret position for inserting text whose direction is counter to the overall line direction.

The `ATSURightwardCursorPosition` function obtains the memory offset for the insertion point to the left of the high caret position, as determined by a move of the specified length at a line direction boundary.

You should use the `ATSULeftwardCursorPosition` function or the function `ATSURightwardCursorPosition` (page 1949) to determine caret position when the user presses the right and left arrow keys.

Except in the case of Indic text (and other cases where the font rearranges the glyphs), for left-to-right text, calling the function `ATSULeftwardCursorPosition` has the same effect as calling `ATSUPreviousCursorPosition` (page 1948). For right-to-left text, calling the function `ATSULeftwardCursorPosition` has the same effect as calling `ATSUNextCursorPosition` (page 1940).

The following code shows how to use the constants `kATSUFromPreviousLayout` and `kATSUFromFollowingLayout` with the function `ATSULeftwardCursorPosition`:

```
typedef struct TLayoutWithEndOffset
{
    ATSUTextLayout      layout;
    UInt32              endOffset;
};

typedef struct TLayoutsWithEndOffsets
{
    UInt32                  count;
    TLayoutWithEndOffset    layouts[];
}
```

```
UniCharArrayOffset MyAbsoluteToRelativeOffset (
                TLayoutsWithEndOffsets * iLayouts,
                UniCharArrayOffset iAbsoluteOffset );
UniCharArrayOffset MyRelativeToAbsoluteOffset (
                TLayoutsWithEndOffsets * iLayouts,
                UInt32 iLayoutIndex,
                UniCharArrayOffset iRelativeOffset );
UniCharArrayOffset MyGetLayoutEndOffset (
                TLayoutsWithEndOffsets * iLayouts,
                UInt32 iLayoutIndex );

/* Passing in current offset relative to the beginning of */
/* the entire text buffer (absolute), */
/* not just the current paragraph. This returns the new (absolute)  */
/* offset relative to the beginning of the entire text buffer.*/
UniCharArrayOffset
MyLeftwardCursorPosition ( TLayoutsWithEndOffsets * iLayouts,
                    UInt32 iLayoutIndex,
                    UniCharArrayOffset iAbsoluteOffset,
                    ATSUCursorMovementType iType )
{
    OSStatus            status;
    UInt32              newLayoutIndex = iLayoutIndex;
    UniCharArrayOffset  newRelativeOffset;

    status = ATSULeftwardCursorPosition(
                iLayouts->layouts[iLayoutIndex].layout,
                MyAbsoluteToRelativeOffset (iLayouts, iAbsoluteOffset  ),
                iType, &newRelativeOffset );

    if ( status == noErr )
    {
        /* If the API returns the same value as */
        /* that passed in then we're at */
        /* the edge of the layout so need to move */
        /* to the adjacent layout. f */
        /* If that value is zero then we're moving to the previous  layout. */
        /* (This is left-to-right text.) */
        if ( (newRelativeOffset == iRelativeOffset) &&
                (iRelativeOffset == 0) )
        {
            /* Don't want to move before the first layout! */
            if ( iLayoutIndex != 0 )
            {
                /* Pass kATSUFromFollowingLayout to the previous  */
                /* ATSUTextLayout. */
                /* Note that the returned offset is relative to
                /* the ATSUTextLayout passed in here.*/
                newLayoutIndex--;
                status = ATSULeftwardCursorPosition(
                            iLayouts[newLayoutIndex],
                            kATSUFromFollowingLayout,
                            iType &newRelativeOffset );
            }
        }
        else
        {
```

```
            UniCharArrayOffset  endAbsoluteOffset = MyGetLayoutEndOffset(
                            iLayouts, iLayoutIndex );

            /* We've moved to the very end of this layout */
            /* (past the trailing carriage return presumably) */
            /* so we're moving to the following layout. */
            /* Make sure we aren't at the */
            /* end of the text buffer. (This is right-to-left text.)  */
            if ( (newRelativeOffset == MyAbsoluteToRelativeOffset  (
                        iLayouts, endAbsoluteOffset )) &&
                       (iLayoutIndex != iLayouts->count) )
            {
                newLayoutIndex++;
                status = ATSULeftwardCursorPosition(
                            iLayouts->layouts[newLayoutIndex],
                            kATSUFromPreviousLayout, iType,
                            &newRelativeOffset );

                /* If we're moving from one paragraph to the following  one */
                /* and we aren't at the beginning of the layout  means
                /* that we're moving to a left-to-right */
                /* paragraph and we must back up one so that*/
                /* we're just before the line ending whitespace
                /* (space or <CR>), unless the */
                /* following layout is the last one. */
                if ( (newRelOffset > 0) && (newLayoutIndex  !=
                         iLayouts->count) )
                    newRelativeOffset--;
            }
        }
    }
    return MyRelativeToAbsoluteOffset( iLayouts, newLayoutIndex,
                        newRelativeOffset );
}
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeDrawing.h

ATSUMatchFontsToText

Examines a text range for characters that cannot be drawn with the current font and suggests a substitute font, if necessary.

```
OSStatus ATSUMatchFontsToText (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iTextStart,
   UniCharCount iTextLength,
   ATSUFontID *oFontID,
   UniCharArrayOffset *oChangedOffset,
   UniCharCount *oChangedLength
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

iTextStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the range to examine. To start at the beginning of the text buffer, pass the constant `kATSUFromTextBeginning`.

iTextLength

A `UniCharCount` value specifying the length of the text range to examine. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

oFontID

A pointer to a `ATSUFontID` value. On return, the value provides a font ID for the suggested substitute font or `kATSUInvalidFontID`, if no substitute font is available.

oChangedOffset

A pointer to a `UniCharArrayOffset` value. On return, this value specifies the offset from the beginning of the text buffer to the first character that cannot be drawn with the current font.

oChangedLength

A pointer to a `UniCharCount` value. On return, this value specifies the length of the text range that cannot be drawn with the current font.

Return Value

A result code. See "ATSUI Result Codes" (page 2068). The result code `noErr` indicates that all the characters in the given range can be rendered with their current font(s) and no font substitution is needed. If you receive either of the result codes `kATSUFontsMatched` or `kATSUFontsNotMatched`, you should update the input range and call `ATSUMatchFontsToText` again to ensure that all the characters in the range can be drawn.

Discussion

When you call the `ATSUMatchFontsToText` function, ATSUI scans the given range of text for characters that cannot be drawn with the currently assigned font. When ATSUI finds such a character, it identifies a substitute font for drawing the character. ATSUI then continues scanning the text range for subsequent characters that cannot be drawn, stopping when it

- finds a character that can be drawn with the currently assigned font, or
- finds a character that cannot be drawn with either the currently assigned font or the substitute font, or
- reaches the end of the text range you have specified

ATSUI's default behavior for finding a substitute font is to recommend the first valid font that it finds when scanning the fonts in the user's system. ATSUI first searches in the standard application fonts for various languages. If that fails, ATSUI searches through the remaining fonts on the system in the order in which the Font Manager returns the fonts. After ATSUI has searched all the fonts in the system, any unmatched text is drawn using the last-resort font. That is, missing glyphs are represented by and empty box to indicate to the

user that a valid font for that character is not installed on their system. You can alter ATSUI's default search behavior by calling the function `ATSUCreateFontFallbacks` (page 1864) and defining your own font fallback settings for the text layout object.

So, for example, if the subrange of text for which you wanted to perform font substitution was the text "abcde", and the characters 'c' and 'd' could not be drawn with the current font, but could be drawn with font X, and the character 'e' either could be drawn with the current font or could not be drawn with font X, then `ATSUMatchFontsToText` produces the ID of font X in the `oFont` parameter and sets the `oChangedOffset` parameter to 2 and the `oChangedLength` parameter to 2.

Because ATSUI does not necessarily completely scan the text range you specify with each call to `ATSUMatchFontsToText`, if ATSUI does find any characters that cannot be rendered with their current font, you should call `ATSUMatchFontsToText` again and update the input range to check that all the subsequent characters in the range can be drawn. For that reason, you should call `ATSUMatchFontsToText` from within a loop to assure that the entire range of text is checked.

Note that calling `ATSUMatchFontsToText` does not cause the suggested font substitution to be performed. If you want ATSUI to perform font substitution for you, you can call the function `ATSUSetTransientFontMatching` (page 1967).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeObjects.h`

ATSUMeasureText

(Deprecated in Mac OS X v10.3. Use `ATSUGetUnjustifiedBounds` (page 1923) instead.)

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUMeasureTextImage

Obtains the image bounding rectangle for a line of text after final layout.

```
OSStatus ATSUMeasureTextImage (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineOffset,
   UniCharCount iLineLength,
   ATSUTextMeasurement iLocationX,
   ATSUTextMeasurement iLocationY,
   Rect *oTextImageRect
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object to examine.

iLineOffset

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the line to examine. To indicate that the specified line starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iLineLength` parameter.

iLineLength

A `UniCharCount` value specifying the length of the text range. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`. However, the image bounds is restricted to the line in which `iLineOffset` resides.

iLocationX

An `ATSUTextMeasurement` value specifying the x-coordinate of the line's origin in the current graphics port or Quartz graphics context. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), for the dimensions of the bounds relative to the current pen location in the current graphics port or graphics context. You can pass `0` to obtain only the dimensions of the bounding rectangle relative to one another, not their actual onscreen position.

iLocationY

An `ATSUTextMeasurement` value specifying the y-coordinate of the line's origin in the current graphics port or Quartz graphics context. Pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), for the dimensions of the bounds relative to the current pen location in the current graphics port or graphics context. You can pass `0` to obtain only the dimensions of the bounding rectangle relative to one another, not their actual onscreen position.

oTextImageRect

A pointer to a `Rect` structure. On return, the structure contains the dimensions of the image bounding rectangle for the text, offset by the values specified in the `iLocationX` and `iLocationY` parameters. If the line is rotated, the sides of the rectangle are parallel to the coordinate axis.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUMeasureTextImage` function obtains the image bounds of a laid-out line of text. These bounds are described by the smallest rectangle that completely encloses the filled or framed parts of a block of text—that is, the text's "inked" glyphs.

In measuring the line, the `ATSUMeasureTextImage` function takes into account line rotation, alignment, and justification, as well as other characteristics that affect layout, such as hanging punctuation. (If the line is rotated, the sides of the rectangle are parallel to the coordinate axes and encompass the rotated line.) If no attributes are set for the line, `ATSUMeasureTextImage` uses the global attributes set for the text layout object.

Because the height of the image bounding rectangle is determined by the actual device metrics, `ATSUMeasureTextImage` ignores any previously set line ascent and descent values for the line it is measuring.

Before calculating the image bounds for the text range, the `ATSUMeasureTextImage` function examines the text layout object to ensure that each of the characters in the range is assigned to a style run. If there are gaps between style runs, `ATSUMeasureTextImage` assigns the characters in the gap to the style run that precedes (in storage order) the gap. If there is no style run at the beginning of the text range, the `ATSUMeasureTextImage` function assigns these characters to the first style run it finds. If there is no style run at the end of the text range, `ATSUMeasureTextImage` assigns the remaining characters to the last style run it finds.

To obtain the final typographic bounds of a line, call the function `ATSUGetGlyphBounds` (page 1904). To calculate the unjustified typographic bounds of a line, call the function `ATSUGetUnjustifiedBounds` (page 1923).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUNextCursorPosition

Obtains the memory offset for the insertion point that follows the current insertion point in storage order, as determined by a move of the specified length.

```
OSStatus ATSUNextCursorPosition (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOldOffset,
   ATSUCursorMovementType iMovementType,
   UniCharArrayOffset *oNewOffset
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value identifying the text layout object to examine.

iOldOffset
A `UniCharArrayOffset` value specifying the memory offset corresponding to the current caret position. To specify the beginning of the text buffer, pass the constant `kATSUFromTextBeginning`.

iMovementType
An `ATSUCursorMovementType` constant identifying the unit of movement. See "Caret Movement Types" (page 2042) for a description of possible values (which range from a single Unicode character to a Unicode word in length). Note that ATSUI may not be able to move the caret by a single Unicode character in some cases, since doing so might place the insertion point in the middle of a surrogate pair.

oNewOffset
A pointer to a `UniCharArrayOffset` value. On return, the value provides the memory offset corresponding to the following insertion point. This offset may be outside the initial text buffer.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUNextCursorPosition` function obtains the memory offset for the insertion point that follows the current insertion point in storage order, as determined by a move of the specified length.

You should use the `ATSUNextCursorPosition` function or the function `ATSUPreviousCursorPosition` (page 1948) to determine caret position when the initial memory offset is not at a line direction boundary. If the initial offset is at a line direction boundary, you should instead use the functions `ATSURightwardCursorPosition` (page 1949) or `ATSULeftwardCursorPosition` (page 1933).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUOffsetToCursorPosition

Obtains the caret position(s) corresponding to a memory offset, after a move of the specified length.

```
OSStatus ATSUOffsetToCursorPosition (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOffset,
   Boolean iIsLeading,
   ATSUCursorMovementType iMovementType,
   ATSUCaret *oMainCaret,
   ATSUCaret *oSecondCaret,
   Boolean *oCaretIsSplit
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value identifying the text layout object to examine.

iOffset
A `UniCharArrayOffset` value specifying the memory offset corresponding to the glyph edge nearest the event, after a movement of the specified type. You can obtain this value by examining the offset produced in the `ioPrimaryOffset` parameter of the function `ATSUPositionToCursorOffset` (page 1945).

iIsLeading
A `Boolean` value indicating whether the specified offset corresponds to the leading or trailing edge of the glyph. You can obtain this information from the function `ATSUPositionToCursorOffset` (page 1945). This value is relevant if the offset occurs at a line direction boundary or within a glyph cluster.

iMovementType
An `ATSUCursorMovementType` constant identifying the unit of cursor movement. See "Caret Movement Types" (page 2042) for a description of possible values (which range from a single Unicode character to a Unicode word in length). Note that ATSUI may not be able to move the cursor by a single Unicode character in some cases, since doing so might place the cursor in the middle of a surrogate pair.

oMainCaret
A pointer to an `ATSUCaret` structure. On return, the structure contains the starting and ending pen locations of the high caret if the value produced in the `oCaretIsSplit` parameter is `true`. If the value is `false`, the structure contains the starting and ending pen locations of the main caret.

oSecondCaret

A pointer to an `ATSUCaret` structure. On return, the structure contains the starting and ending pen locations of the low caret if the value passed back in the `oCaretIsSplit` parameter is `true`. If the value is `false`, the structure contains the starting and ending pen locations of the main caret (that is, the same values as the `oMainCaret` parameter).

oCaretIsSplit

A pointer to a `Boolean` value. On return, the value indicates whether the offset specified in the `iOffset` parameter occurs at a line direction boundary. If `true`, the offset occurs at a line direction boundary; otherwise, `false`.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUOffsetToPosition

Obtains the caret position(s) corresponding to a memory offset.

```
OSStatus ATSUOffsetToPosition (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOffset,
   Boolean iIsLeading,
   ATSUCaret *oMainCaret,
   ATSUCaret *oSecondCaret,
   Boolean *oCaretIsSplit
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object to examine.

iOffset

A `UniCharArrayOffset` value specifying the memory offset for which to obtain the corresponding caret position. To respond to a mouse-down event, pass the offset produced in the `ioPrimaryOffset` parameter of the function `ATSUPositionToOffset` (page 1946)—that is, the offset corresponding to the glyph edge closest to the event.

iIsLeading

A `Boolean` value indicating whether the offset corresponds to the leading or trailing edge of the glyph. You can obtain this information from the function `ATSUPositionToOffset` (page 1946). This value is relevant if the offset occurs at a line direction boundary or within a glyph cluster.

oMainCaret

A pointer to an `ATSUCaret` structure. On return, the structure contains the starting and ending pen locations of the high caret if the value produced in `oCaretIsSplit` is `true`. If the value is `false`, the structure contains the starting and ending pen locations of the main caret.

oSecondCaret

A pointer to an `ATSUCaret` structure. On return, the structure contains the starting and ending pen locations of the low caret if the value passed back in the `oCaretIsSplit` parameter is `true`. If the value is `false`, the structure contains the starting and ending pen locations of the main caret (that is, the same values as the `oMainCaret` parameter).

oCaretIsSplit

A pointer to a `Boolean` value. On return, the value indicates whether the offset specified in the `iOffset` parameter occurs at a line direction boundary. If `true`, the offset occurs at a line direction boundary; otherwise, `false`.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The process of hit-testing text obtains the location of a mouse-down event relative both to the position of onscreen glyphs and to the corresponding offset between character codes in memory. You can then use the location information obtained by hit-testing to set the insertion point (that is, the caret) or selection range (for highlighting).

Hit-testing text is complicated by the fact that a given line of text may be bidirectional. Therefore, the onscreen order of glyphs may not readily correspond to the storage order of the corresponding character codes. And the concept of which glyph comes "first" in a line of text cannot always be limited to the visual terms "left" and "right." Because of these complexities, it is more accurate to speak in terms of "leading" and "trailing" edges to glyphs. A "leading edge" is defined as the edge of a glyph that you first encounter when you read the text that includes that glyph. For example, when reading Roman text, you first encounter the left edge of a Roman glyph. Similarly, the "trailing edge" is defined as the edge of the glyph encountered last.

ATSUI can translate the location of a mouse click into an onscreen position, as well as to a memory offset. When you use ATSUI for hit-testing, ATSUI takes into account the glyph edge (whether leading or trailing) nearest to where the click occurred, thus providing positional information in complex situations, such as at line direction boundaries or within glyph clusters.

Line direction boundaries can occur on the trailing edges of two glyphs, the leading edges of two glyphs, or at the beginning or end of a text segment. At direction boundaries, a single insertion point in memory can require two caret positions onscreen, one for text entry in each direction. The two separate carets (known as a split caret or a dual caret) consist of a high caret and a low caret. The high (primary) caret is displayed at the caret position for inserting text whose direction corresponds to the line direction (the dominant direction for the overall line of text). The low (secondary) caret is displayed at the caret position for inserting text whose direction is counter to the overall line direction.

The first step in obtaining the caret position(s) for a mouse-down event is to pass the location (in local coordinates, relative to the line origin) of the event to the function `ATSUPositionToOffset` (page 1946). The `ATSUPositionToOffset` function produces the memory offset corresponding to the glyph edge nearest the event. If the mouse-down event occurs at a line direction boundary or within a glyph cluster, the `ATSUPositionToOffset` function produces two offsets. You can then provide the offset(s) to the `ATSUOffsetToPosition` function, to obtain the actual caret position(s) for the event.

The `ATSUOffsetToPosition` function produces two structures of type `ATSUCaret`. These structures contain the pen positioning information needed to draw the caret(s) for the event, specified relative to the origin of the line in the current graphics port or graphics context. Specifically, the `ATSUCaret` structures contain x-y coordinates for both the caret's starting and ending pen positions (the latter taking into account line rotation, caret slanting, and split-caret appearances).

If the memory offset you pass to `ATSUOffsetToPosition` is at a line boundary, the structure produced in the `oMainCaret` parameter contains the starting and ending pen locations for the high caret, while the `oSecondCaret` parameter contains the corresponding values for the low caret. If the offset is not at a line boundary, both parameters contain the starting and ending pen locations of the main caret.

Because you provide the `ATSUOffsetToPosition` function an offset relative to the origin of the line where the hit occurred, `ATSUOffsetToPosition` produces positioning information that is also relative. Therefore, you must transform the positions produced by the `ATSUOffsetToPosition` function before drawing the caret(s). To transform the caret location(s), add the starting and ending caret coordinates to the coordinates of the origin of the line in which the hit occurred. For example, if `ATSUOffsetToPosition` produces starting and ending pen locations of (25,0), (25,25) in the `oMainCaret` parameter (and the `oSecondCaret` parameter contains the same coordinates, meaning that the caret was not split), you would add these to the position of the origin of the line in the graphics port or context. If the position of the line origin was at (50,50), then the starting and ending pen locations of the caret would be (75,50), (75,75).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUOverwriteAttributes

Copies to a destination style object the nondefault style attribute settings of a source style object.

```
OSStatus ATSUOverwriteAttributes (
   ATSUStyle iSourceStyle,
   ATSUStyle iDestinationStyle
);
```

Parameters

`iSourceStyle`
An `ATSUStyle` value specifying the style object from which to copy nondefault style attributes.

`iDestinationStyle`
An `ATSUStyle` value specifying the style object containing the style attributes to be overwritten.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUOverwriteAttributes` function copies all nondefault style attribute values from a source style object to a destination style object. The source object's nondefault values are applied to the destination object whether or not the destination object also has nondefault values for the copied attributes. All other settings in the destination style object are left unchanged.

`ATSUOverwriteAttributes` does not copy the contents of memory referenced by pointers within custom style attributes or within reference constants. You are responsible for ensuring that this memory remains valid until both the source and destination style objects are disposed of.

To create a style object that contains all the contents of another style object, call the function `ATSUCreateAndCopyStyle` (page 1863). To copy all the style attributes (including any default settings) of a style object into an existing style object, call the function `ATSUCopyAttributes` (page 1856). To copy style attributes that are set in the source but not in the destination style object, call the function `ATSUUnderwriteAttributes` (page 1972).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUPositionToCursorOffset

Obtains the memory offset for the glyph edge nearest a mouse-down event, after a move of the specified length.

```
OSStatus ATSUPositionToCursorOffset (
   ATSUTextLayout iTextLayout,
   ATSUTextMeasurement iLocationX,
   ATSUTextMeasurement iLocationY,
   ATSUCursorMovementType iMovementType,
   UniCharArrayOffset *ioPrimaryOffset,
   Boolean *oIsLeading,
   UniCharArrayOffset *oSecondaryOffset
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value identifying the text layout object in which the mouse-down event occurred.

iLocationX
An `ATSUTextMeasurement` value specifying the x-coordinate of the event, in local coordinates, relative to the origin of the line where the event occurred. That is, to specify the x-coordinate value, you should subtract the x-coordinate of the line origin from the x-coordinate of the event (in local coordinates). You can pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), for the location of the mouse-down event relative to the current pen location in the current graphics port.

iLocationY
An `ATSUTextMeasurement` value specifying the y-coordinate of the event, in local coordinates, relative to the origin of the line where the event occurred. That is, to specify the y-coordinate value, you should subtract the y-coordinate of the line origin from the y-coordinate of the event (in local coordinates). You can pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), for the location of the mouse-down event relative to the current pen location in the current graphics port.

iMovementType
An `ATSUCursorMovementType` constant identifying the unit of movement. See "Caret Movement Types" (page 2042) for a description of possible values (which range from a single Unicode character to a Unicode word in length). Note that ATSUI may not be able to move the caret by a single Unicode character in some cases, since doing so might place the insertion point in the middle of a surrogate pair.

ioPrimaryOffset

A pointer to a `UniCharArrayOffset` value specifying the offset corresponding to the beginning of the line where the event occurred. On return, the value specifies the offset corresponding to the glyph edge nearest the event, after a movement of the specified type. This offset corresponds to where the insertion point would be placed after the move. To determine whether this offset indicates the leading or trailing edge of the glyph, you can examine the value produced in the *oIsLeading* parameter.

oIsLeading

A pointer to a `Boolean` value. On return, the value indicates whether the offset produced in the `ioPrimaryOffset` parameter is leading or trailing. The `ATSUPositionToOffset` function produces a value of `true` if the offset is leading (that is, more closely associated with the subsequent character in memory). It produces a value of `false` if the offset is trailing (that is, more closely associated with the preceding character in memory).

oSecondaryOffset

A pointer to a `UniCharArrayOffset` value. On return, the value typically specifies the same offset as that produced in the *ioPrimaryOffset* parameter, unless the event occurred within a glyph cluster or at a line direction boundary. If so, the value specifies the secondary offset, for the glyph edge furthest from the event.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUPositionToCursorOffset` function produces the memory offset for the glyph edge nearest a mouse-down event, after a move of the specified length. This offset corresponds to where an insertion point would be placed after the move.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUPositionToOffset

Obtains the memory offset for the glyph edge nearest a mouse-down event.

```
OSStatus ATSUPositionToOffset (
   ATSUTextLayout iTextLayout,
   ATSUTextMeasurement iLocationX,
   ATSUTextMeasurement iLocationY,
   UniCharArrayOffset *ioPrimaryOffset,
   Boolean *oIsLeading,
   UniCharArrayOffset *oSecondaryOffset
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object in which the mouse-down event occurred.

iLocationX

An `ATSUTextMeasurement` value specifying the x-coordinate of the event, in local coordinates, relative to the origin of the line where the event occurred. That is, to specify the x-coordinate value, you should subtract the x-coordinate of the line origin from the x-coordinate of the hit point (in local coordinates). You can pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), for the location of the mouse-down event relative to the current pen location in the current graphics port.

iLocationY

An `ATSUTextMeasurement` value specifying the y-coordinate of the event, in local coordinates, relative to the origin of the line where the event occurred. That is, to specify the y-coordinate value, you should subtract the y-coordinate of the line origin from the y-coordinate of the hit point (in local coordinates). You can pass the constant `kATSUUseGrafPortPenLoc`, described in "Convenience Constants" (page 2043), for the location of the mouse-down event relative to the current pen location in the current graphics port.

ioPrimaryOffset

A pointer to a `UniCharArrayOffset` value specifying the offset corresponding to the beginning of the line where the event occurred. On return, the value specifies the offset corresponding to the glyph edge that is visually closest to the event. To determine whether this offset indicates the leading or trailing edge of the glyph, you can examine the value produced in the *oIsLeading* parameter.

oIsLeading

A pointer to a `Boolean` value. On return, the value indicates whether the offset produced in the `ioPrimaryOffset` parameter is leading or trailing. The function `ATSUPositionToOffset` produces a value of `true` if the offset is leading (that is, more closely associated with the subsequent character in memory). It produces a value of `false` if the offset is trailing (that is, more closely associated with the preceding character in memory).

oSecondaryOffset

A pointer to a `UniCharArrayOffset` value. On return, the value typically specifies the same offset as that produced in the *ioPrimaryOffset* parameter, unless the event occurred within a glyph cluster or at a line direction boundary. If so, the value specifies a secondary offset. The secondary offset is associated with the glyph that has a different direction from the primary line direction.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The process of hit-testing text obtains the location of a mouse-down event relative both to the position of onscreen glyphs and to the corresponding offset between character codes in memory. You can then use the location information obtained by hit-testing to set the insertion point (that is, the caret) or selection range (for highlighting).

Hit-testing text is complicated by the fact that a given line of text may be bidirectional. Therefore, the onscreen order of glyphs may not readily correspond to the storage order of the corresponding character codes. And the concept of which glyph comes "first" in a line of text cannot always be limited to the visual terms "left" and "right." Because of these complexities, it is more accurate to speak in terms of "leading" and "trailing" edges to glyphs. A "leading edge" is defined as the edge of a glyph that you first encounter when you read the text that includes that glyph. For example, when reading Roman text, you first encounter the left edge of a Roman glyph. Similarly, the "trailing edge" is defined as the edge of the glyph encountered last.

ATSUI can translate the location of a mouse click into an onscreen position, as well as to a memory offset. When you use ATSUI for hit-testing, ATSUI takes into account the glyph edge (whether leading or trailing) nearest to where the click occurred, thus providing positional information in complex situations, such as at line direction boundaries or within glyph clusters.

The first step in obtaining the caret position(s) for a mouse-down event is to pass the location (in local coordinates, relative to the line origin) of the event to the function `ATSUPositionToOffset`. For example, if you have a mouse-down event whose position in local coordinates is (75,50), you would subtract this value from the position of the origin of the line in the current graphics port. If the position of the origin of the line in the current graphics port is (50,50), then the relative position of the event that you would pass in the `iLocationX` and `iLocationY` parameters is (25,0).

The `ATSUPositionToOffset` function produces the memory offset corresponding to the glyph edge nearest the event. If the mouse-down event occurs at a line direction boundary or within a glyph cluster, `ATSUPositionToOffset` produces two offsets. You can then provide the offset(s) to the `ATSUOffsetToPosition` (page 1942) function, to obtain the actual caret position(s) for the event.

When you call the `ATSUPositionToOffset` function, ATSUI examines the Unicode directionality of the character corresponding to the event location. The `ATSUPositionToOffset` function produces a value of `true` in the *`oIsLeading`* parameter if the offset is leading (that is, more closely associated with the subsequent character in memory and therefore indicative of a left-to-right line direction). It produces a value of `false` if the offset is trailing (that is, more closely associated with the preceding character in memory and indicative of a right-to-left line direction).

Finally, note that when the event occurs beyond the leftmost or rightmost caret positions of the line (not taking into account line rotation), such that no glyph corresponds to the location of the hit, the `ATSUPositionToOffset` function produces the primary offset of the closest edge of the line to the input location. The `oIsLeading` flag depends on the directionality of the closest glyph and the side of the line to which the input location is closest. In this case, the secondary offset is equal to the primary offset, since no glyph was hit.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeDrawing.h`

ATSUPreviousCursorPosition

Obtains the memory offset for the insertion point that precedes the current insertion point in storage order, as determined by a move of the specified length.

```
OSStatus ATSUPreviousCursorPosition (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOldOffset,
   ATSUCursorMovementType iMovementType,
   UniCharArrayOffset *oNewOffset
);
```

Parameters

`iTextLayout`

An `ATSUTextLayout` value identifying the text layout object to examine.

`iOldOffset`

A `UniCharArrayOffset` value specifying the memory offset corresponding to the current caret position. To specify the beginning of the text buffer, pass the constant `kATSUFromTextBeginning`,

iMovementType

An `ATSUCursorMovementType` constant identifying the unit of movement. See "Caret Movement Types" (page 2042) for a description of possible values (which range from a single Unicode character to a Unicode word in length). Note that ATSUI may not be able to move the caret by a single Unicode character in some cases, since doing so might place the insertion point in the middle of a surrogate pair.

oNewOffset

A pointer to a `UniCharArrayOffset` value. On return, the value provides the memory offset corresponding to the preceding insertion point. This offset may be outside the initial text buffer.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUPreviousCursorPosition` function obtains the memory offset for the insertion point that precedes the current insertion point in storage order, as determined by a move of the specified length.

You should use the `ATSUPreviousCursorPosition` function or the function `ATSUNextCursorPosition` (page 1940) to determine caret position when the initial offset is not at a line direction boundary. If the initial offset is at a line direction boundary, you should instead use the functions `ATSURightwardCursorPosition` (page 1949) or `ATSULeftwardCursorPosition` (page 1933).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSURightwardCursorPosition

Obtains the memory offset for the insertion point to the right of the high caret position, as determined by a move of the specified length at a line direction boundary.

```
OSStatus ATSURightwardCursorPosition (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iOldOffset,
   ATSUCursorMovementType iMovementType,
   UniCharArrayOffset *oNewOffset
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object to examine.

iOldOffset

A `UniCharArrayOffset` value specifying the memory offset corresponding to the current caret position. To specify the beginning of the text buffer, pass the constant `kATSUFromTextBeginning`. For bidirectional text, you can specify the previous layout by passing the constant `kATSUFromPreviousLayout` and the following layout by passing the constant `kATSUFromFollowingLayout`. See the Discussion for the function `ATSULeftwardCursorPosition` (page 1933) for an example of how these constants can be used.

iMovementType

An `ATSUCursorMovementType` constant identifying the unit of movement. See "Caret Movement Types" (page 2042) for a description of possible values (which range from a single Unicode character to a Unicode word in length). Note that ATSUI may not be able to move the caret by a single Unicode character in some cases, since doing so might place the insertion point in the middle of a surrogate pair.

oNewOffset

A pointer to a `UniCharArrayOffset` value. On return, the value provides the memory offset corresponding to the new insertion point. This offset may be outside the initial text buffer.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
Line direction boundaries can occur on the trailing edges of two glyphs, the leading edges of two glyphs, or at the beginning or end of a text segment. At direction boundaries, a single insertion point in memory can require two caret positions onscreen, one for text entry in each direction. The two separate carets (known as a split caret or a dual caret) consist of a high caret and a low caret. The high (primary) caret is displayed at the caret position for inserting text whose direction corresponds to the line direction (the dominant direction for the overall line of text). The low (secondary) caret is displayed at the caret position for inserting text whose direction is counter to the overall line direction.

The `ATSURightwardCursorPosition` function obtains the memory offset for the insertion point to the right of the high caret position, as determined by a move of the specified length at a line direction boundary.

You should use the `ATSURightwardCursorPosition` function or the function `ATSULeftwardCursorPosition` (page 1933) to determine caret position when the user presses the right and left arrow keys.

Except in the case of Indic text (and other cases where the font rearranges the glyphs), for left-to-right text, calling the function `ATSURightwardCursorPosition` has the same effect as calling `ATSUNextCursorPosition` (page 1940). For right-to-left text, calling the function `ATSURightwardCursorPosition` has the same effect as calling `ATSUPreviousCursorPosition` (page 1948).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUSetAttributes

Sets style attribute values in a style object.

```
OSStatus ATSUSetAttributes (
   ATSUStyle iStyle,
   ItemCount iAttributeCount,
   const ATSUAttributeTag iTag[],
   const ByteCount iValueSize[],
   const ATSUAttributeValuePtr iValue[]
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to set attributes.

iAttributeCount

An `ItemCount` value specifying the number of attributes to set. This value should correspond to the number of elements in the `iTag` and `iValueSize` arrays.

iTag

A pointer to the initial `ATSUAttributeTag` value in an array of attribute tags. Each element in the array must contain a valid style attribute tag that corresponds to the style attribute value to set. Note that an attribute tag cannot be used in versions of the Mac OS that are earlier than the version in which the tag was introduced. For example, a tag available in Mac OS version 10.2 cannot be used in Mac OS version 10.1 or earlier. You can call the function Gestalt to check version information for ATSUI. See "Attribute Tags" (page 2030) for a description of the Apple-defined style attribute tag constants and for availability information.

iValueSize

A pointer to the initial `ByteCount` value in an array of attribute value sizes. Each element in the array must contain the size (in bytes) of the corresponding style run attribute value being set. `ATSUSetAttributes` sets style attributes after confirming the sizes in the array.

iValue

A pointer to the initial `ATSUAttributeValuePtr` value in an array of attribute value pointers. Each pointer in the array must reference an attribute value corresponding to a tag in the `iTag` array. The value referenced by the pointer must be legal for that tag.

Return Value

A result code. See "ATSUI Result Codes" (page 2068). If there is a function error, `ATSUSetAttributes` does not set any attributes in the style object.

Discussion

Style attributes are a collection of values and settings that override the font-specified behavior for displaying and formatting text in a style run. To specify a style attribute, ATSUI uses a "triple" consisting of (1) an attribute tag, (2) a value for that tag, and (3) the size of the value.

The `ATSUSetAttributes` function enables you to set multiple style attribute values for a style object. When you call `ATSUSetAttributes`, any style attributes that you do not set retain their previous values. To set font features and font variations, call the functions `ATSUSetFontFeatures` (page 1952) and `ATSUSetVariations` (page 1967), respectively.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSUnicodeObjects.h`

ATSUSetFontFallbacks

Sets, on a global scope, the font list and search order for ATSUI to use when a font does not have the glyph needed to image a character. (Deprecated in Mac OS X v10.3. Use font fallback objects instead.)

Not recommended.

```
OSStatus ATSUSetFontFallbacks (
   ItemCount iFontFallbacksCount,
   const ATSUFontID iFontIDs[],
   ATSUFontFallbackMethod iFontFallbackMethod
);
```

Parameters

iFontFallbacksCount

An `ItemCount` value specifying the number of fonts to be searched. This value should be equivalent to the number of elements in the `iFontIDs` array.

iFontIDs

A pointer to the first `ATSUFontID` value in the array of fonts to be searched.

iFontFallbackMethod

An `ATSUFontFallbackMethod` value specifying the order in which ATSUI is to search the fonts. See "Font Fallback Methods" (page 2048) for a description of possible search orders.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

When you call `ATSUSetFontFallbacks`, any settings you apply are global to the process and are used by all ATSUI clients in the process. Therefore, any ATSUI clients in the process can change these global font fallback settings unexpectedly. Other application threads can modify the font fallbacks settings, as well. The only way to ensure that ATSUI uses your preferred font fallback settings for your text is to create a font fallback object and associated it with a text layout object.

You create a font fallback object by calling the function `ATSUCreateFontFallbacks` (page 1864). You define settings for the object by calling the function `ATSUSetObjFontFallbacks` (page 1958). To associate the font fallback object with a text layout object, call either of the functions `ATSUSetLayoutControls` (page 1955) or `ATSUSetLineControls` (page 1956). See *Inside Mac OS X: Rendering Unicode Text With ATSUI* for step-by-step instructions on creating a font fallback object and associating it with a text layout object.

Special Considerations

You should not use this function because it operates on a global scope and may not be available in future versions of ATSUI. Instead, use font fallback objects as described in the Discussion.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUSetFontFeatures

Sets font features in a style object.

```
OSStatus ATSUSetFontFeatures (
   ATSUStyle iStyle,
   ItemCount iFeatureCount,
   const ATSUFontFeatureType iType[],
   const ATSUFontFeatureSelector iSelector[]
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to set font features.

iFeatureCount

An `ItemCount` value specifying the number of font features to set. This value should correspond to the number of elements in the `iType` and `iSelector` arrays.

iType

A pointer to the initial `ATSUFontFeatureType` value in an array of feature types. Each element in the array must contain a valid feature type that corresponds to a feature selector in the `iSelector` array. To obtain the valid feature types for a font, call the function `ATSUGetFontFeatureTypes` (page 1898).

iSelector

A pointer to the initial `ATSUFontFeatureSelector` value in an array of feature selectors. Each element in the array must contain a valid feature selector that corresponds to a feature type in the `iType` array. To obtain the valid feature selectors for a font, call the function `ATSUGetFontFeatureSelectors` (page 1897).

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUSetFontFeatures` function enables you to set multiple font features for a style object. Any unset font features retain their font-defined default values. To set style attributes and font variations for a style object, call the functions `ATSUSetAttributes` (page 1950) and `ATSUSetVariations` (page 1967), respectively.

The constants that represent font feature types are defined in the header file `SFNTLayoutTypes.h`. When you use ATSUI to access and set font features, you must use the constants defined in this header file, which are described in *Inside Mac OS X: Rendering Unicode Text With ATSUI*. As feature types can be added at any time, you should check Apple's font feature registry website for the most up-to-date list of font feature types and selectors: http://developer.apple.com/fonts/Registry/index.html.

Version Notes

Prior to ATSUI 1.2, `ATSUSetFontFeatures` does not remove contradictory font features. You are responsible for maintaining your own list and removing contradictory settings when they occur. Beginning with ATSUI 1.2, `ATSUSetFontFeatures` removes contradictory font features if they are set.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSUnicodeFonts.h`

ATSUSetHighlightingMethod

Sets the method ATSUI uses to highlight and unhighlight text for a text layout object.

```
OSStatus ATSUSetHighlightingMethod (
   ATSUTextLayout iTextLayout,
   ATSUHighlightMethod iMethod,
   const ATSUUnhighlightData *iUnhighlightData
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value identifying the text layout object for which to set the highlighting method.

iMethod

An `ATSUHighlightMethod` value specifying the type of highlighting for ATSUI to use (`kInvertHighlighting` or `kRedrawHighlighting`). The default highlighting method, if you do not call `ATSUSetHighlightingMethod`, is inversion. See "Highlight Methods" (page 2053) for a description of available values.

iUnhighlightData

A pointer to an `ATSUUnhighlightData` structure if you are setting the *iMethod* parameter to `kRedrawHighlighting` or `NULL` if setting `iMethod` to `kInvertHighlighting`. Before calling `ATSUSetHighlightingMethod`, you should set the `ATSUUnhighlightData` structure to contain the data needed (either a color or a UPP for a background drawing callback) to redraw the background.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

In Mac OS 9 and by default in Mac OS X (except with Cocoa applications—see below), ATSUI highlights text by "inverting" the region containing the text, that is, its background color. Although inversion provides satisfactory highlighting in most cases, it does not always provide the best result for grayscale text. (Mac OS X sets a lower threshold for antialiasing, while in Mac OS 9 grayscale text can be turned off by the user.)

In Mac OS X, when using a Quartz graphics context, you can instruct ATSUI to use the redraw method of highlighting, rather than simple inversion. (Note that Cocoa applications always use the redraw method of highlighting.) The redraw method allows for accurate highlighting of more complex backgrounds, such as those containing multiple colors, patterns, or pictures. To set redrawing on, call the `ATSUSetHighlightingMethod` function and specify that the redraw method be used (by passing `kRedrawHighlighting` in the *iMethod* parameter).

If you specify the redraw method of highlighting when you call `ATSUSetHighlightingMethod`, then you must also specify how the background is to be redrawn when the function `ATSUUnhighlightText` (page 1974) is called. ATSUI can restore the desired background in one of two ways, depending on the background's complexity:

- When the background is a single color (such as white), ATSUI can readily unhighlight the background. In such a case, you specify the background color that ATSUI uses by calling `ATSUSetHighlightingMethod` and setting `iUnhighlightData.dataType` to `kATSUBackgroundColor` and providing the background color in `iUnhighlightData.unhighlightData`. With these settings defined, when you call `ATSUUnhighlightText`, ATSUI simply calculates the previously highlighted area, repaints it with the specified background color, and then redraws the text.
- When the background is more complex (containing, for example, multiple colors, patterns, or pictures), you must provide a redraw background callback function when you call `ATSUSetHighlightingMethod`. You do this by setting `iUnhighlightData.dataType` to `kATSUBackgroundCallback` and providing a `RedrawBackgroundUPP` in `iUnhighlightData.unhighlightData`. Then when you call `ATSUUnhighlightText` and ATSUI calls your callback, you are responsible for redrawing the background

of the unhighlighted area. If you choose to also redraw the text, then your callback should return `false` as a function result. If your callback returns `true` ATSUI redraws any text that needs to be redrawn. See `RedrawBackgroundProcPtr` (page 1999) for additional information.

Version Notes
Mac OS 9 applications cannot use the redraw method of highlighting and must use the inversion method, instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUSetLayoutControls

Sets layout control attribute values in a text layout object.

```
OSStatus ATSUSetLayoutControls (
   ATSUTextLayout iTextLayout,
   ItemCount iAttributeCount,
   const ATSUAttributeTag iTag[],
   const ByteCount iValueSize[],
   const ATSUAttributeValuePtr iValue[]
);
```

Parameters

`iTextLayout`
An `ATSUTextLayout` value specifying the text layout object for which to set layout control attributes.

`iAttributeCount`
An `ItemCount` value specifying the number of attributes to set. This value should correspond to the number of elements in the `iTag` and `iValueSize` arrays.

`iTag`
A pointer to the initial `ATSUAttributeTag` value in an array of layout control attribute tags. Each element in the array must contain a valid tag that corresponds to the layout control attribute to set. See "Attribute Tags" (page 2030) for a description of the Apple-defined layout control attribute tag constants.

`iValueSize`
A pointer to the initial `ByteCount` value in an array of attribute value sizes. Each element in the array must contain the size (in bytes) of the corresponding layout control attribute being set. `ATSUSetLayoutControls` sets layout attributes after confirming the sizes in the array.

`iValue`
A pointer to the initial `ATSUAttributeValuePtr` value in an array of attribute value pointers. Each value in the array must correspond to a tag in the `iTag` array and be a legal value for that tag.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
When you use ATSUI to image your text, you can control the text's display and formatting at a number of different levels.

One level is that of the entire text range associated with your text layout object, also known as the "layout level." To affect display and formatting on this level, you can specify various layout control attributes using the `ATSUSetLayoutControls` function. These attributes affect the width of the text area from margin to margin, the alignment of the text, its justification, rotation, and direction, as well as other layout options.

Another level is that of a single line of text, that is, the "line level." To affect display and formatting on this level, you specify various line control attributes via the function `ATSUSetLineControls` (page 1956). These attributes are similar to those that you can apply on a full-layout basis, but each affects only an individual text line.

Given that ATSUI allows you to control similar aspects of the display and formatting of your text at either the line level or the layout level (or both, or neither), it is up to you to decide how much layout control to take. However, you should note the following:

- Setting layout control attributes overrides the corresponding default layout-level settings for a text layout object. Any layout attributes that you do not set retain the default values described in "Attribute Tags" (page 2030).
- Setting line control attributes overrides the corresponding layout-level settings (whether set or at default values) for a text layout object. This is true even if you set the layout-level attributes subsequently to the line-level ones.
- From a performance standpoint, it is preferable to work from the layout level and not specify layout line by line unless necessary.

Finally, it is also possible to control the display and formatting of your text at the level of an individual character or "run" of characters. At this level, you customize layout by manipulating style settings in a style object. Among the character-level aspects you can control are style attributes (such as font size and color), font features (such as ligatures), and font variations (such as continually varying font weights or widths). However, there are certain line control attributes (specified via the `ATSLineLayoutOptions` flags) that can override style attributes applied to the same text.

Similarly to style attributes, you use a "triple" to specify a line or layout control attribute. That is, an attribute tag, the value of the attribute it sets, and the size (in bytes) of the attribute value. Attribute tags are constants supplied by ATSUI. Attribute values may be a scalar, a structure, or a pointer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeObjects.h`

ATSUSetLineControls

Sets layout control attribute values for a single line in a text layout object.

```
OSStatus ATSUSetLineControls (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineStart,
   ItemCount iAttributeCount,
   const ATSUAttributeTag iTag[],
   const ByteCount iValueSize[],
   const ATSUAttributeValuePtr iValue[]
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object for which to set line control attribute values.

iLineStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the first character of the line for which to set control attributes.

iAttributeCount

An `ItemCount` value specifying the number of attributes to set. This value should correspond to the number of elements in the `iTag` and `iValueSize` arrays.

iTag

A pointer to the initial `ATSUAttributeTag` value in an array of line control attribute tags. Each element in the array must contain a valid tag that corresponds to the line control attribute to set. See "Attribute Tags" (page 2030) for a description of the Apple-defined line control attribute tag constants.

iValueSize

A pointer to the initial `ByteCount` value in an array of attribute value sizes. Each element in the array must contain the size (in bytes) of the corresponding line control attribute being set. `ATSUSetLineControls` sets line attributes after confirming the sizes in the array.

iValue

A pointer to the initial `ATSUAttributeValuePtr` value in an array of attribute value pointers. Each value in the array must correspond to a tag in the `iTag` array and be a legal value for that tag.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

When you use ATSUI to image your text, you can control the text's display and formatting at a number of different levels. One level is that of the entire text range associated with your text layout object, also known as the "layout level." To affect display and formatting on this level, you can specify various layout control attributes using the `ATSUSetLayoutControls` (page 1955) function. These attributes affect the width of the text area from margin to margin, the alignment of the text, its justification, rotation, and direction, as well as other layout options.

Another level is that of a single line of text, that is, the "line level." To affect display and formatting on the line level, you specify various line control attributes using the function `ATSUSetLineControls`. These attributes are similar to those that you can apply on a full-layout basis, but each affects only an individual text line.

You can break text into lines by calling the functions `ATSUBatchBreakLines` (page 1844) or `ATSUBreakLine` (page 1846). You can define separate lines of text by specifying soft breaks either by

- calling the function `ATSUBatchBreakLines`
- calling the function `ATSUBreakLine` with the `iUseAsSoftBreak` parameter set to `true`

- specifying the soft line breaks using the function `ATSUSetSoftLineBreak`

Given that ATSUI allows you to control similar aspects of the display and formatting of your text at either the line level or the layout level (or both, or neither), it is up to you to decide how much layout control to take. However, you should note the following:

- Setting layout control attributes overrides the corresponding default layout-level settings for a text layout object. Any layout attributes that you do not set retain the default values described in "Attribute Tags" (page 2030).
- Setting line control attributes overrides the corresponding layout-level settings (whether set or at default values) for a text layout object. This is true even if you set the layout-level attributes subsequently to the line-level ones. Any line attributes that you do not set retain their default values.
- From a performance standpoint, it is preferable to work from the layout level and not specify layout line by line unless necessary.

Finally, it is also possible to control the display and formatting of your text at the level of an individual character or "run" of characters. At this level, you customize layout by manipulating style settings in a style object. Among the character-level aspects you can control are style attributes (such as font size and color), font features (such as ligatures), and font variations (such as continually varying font weights or widths). However, there are certain line control attributes (specified via the `ATSLineLayoutOptions` flags) that can override style attributes applied to the same text.

Similarly to style attributes, you use a "triple" to specify a line or layout control attribute. That is, an attribute tag, the value of the attribute it sets, and the size (in bytes) of the attribute value. Attribute tags are constants supplied by ATSUI. Attribute values may be a scalar, a structure, or a pointer.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUSetObjFontFallbacks

Assigns a font list and a font-search method to a font fallback object.

```
OSStatus ATSUSetObjFontFallbacks (
   ATSUFontFallbacks iFontFallbacks,
   ItemCount iFontFallbacksCount,
   const ATSUFontID iFonts[],
   ATSUFontFallbackMethod iFontFallbackMethod
);
```

Parameters

`iFontFallbacks`
An `ATSUFontFallbacks` value specifying the font fallback object for which to define settings.

`iFontFallbacksCount`
An `ItemCount` value specifying the number of fonts that ATSUI is to search. This value is typically equal to the number of font IDs you are providing in the *`iFonts`* array.

iFonts

A pointer to the first `ATSUFontID` value in an array of font IDs identifying the fonts ATSUI is to search.

iFontFallbackMethod

An `ATSUFontFallbackMethod` value identifying the order in which ATSUI is to search. See "Font Fallback Methods" (page 2048) for a description of possible values.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUSetObjFontFallbacks` function defines the settings for a font fallback object. These settings determine the font list and search order that ATSUI uses when seeking substitute fonts for the text layout object with which the font fallback object is associated.

Creating, defining settings for, and associating a font fallback object with a text layout object is the only way to ensure that ATSUI uses your preferred font fallback settings for your text. To create a font fallback object, you first call the function `ATSUCreateFontFallbacks` (page 1864). You then define settings for the object by calling the `ATSUSetObjFontFallbacks` function. To associate the font fallback object with a text layout object call the function `ATSUSetLayoutControls` (page 1955). You pass these functions the control attribute value `kATSULineFontFallbacksTag` to set the font fallback object.

If you do not call `ATSUSetObjFontFallbacks` to change ATSUI's default search behavior, ATSUI searches all the fonts on the system sequentially and uses the first valid font it finds for a substitute. If you are careful in ordering the fonts that you supply to `ATSUSetObjFontFallbacks`, you can minimize the time ATSUI needs to find a substitute font.

Font fallback settings affect the behavior of the function `ATSUMatchFontsToText` (page 1936) and of font selection during layout and drawing when the function `ATSUSetTransientFontMatching` (page 1967) is set to on.

To obtain the font list and font-search method associated with a font fallback object, call the function `ATSUGetObjFontFallbacks` (page 1914).

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUSetRunStyle

Defines a style run by associating style information with a run of text.

```
OSStatus ATSUSetRunStyle (
   ATSUTextLayout iTextLayout,
   ATSUStyle iStyle,
   UniCharArrayOffset iRunStart,
   UniCharCount iRunLength
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying a text layout object with an associated text buffer. `ATSUSetRunStyle` assigns a style object to a run of text in this buffer.

iStyle

An `ATSUStyle` value specifying the style object to associate with the text run.

iRunStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the text run.

iRunLength

A `UniCharCount` value specifying the length of the text run.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

A text run consists of one or more characters that are contiguous in memory. If you associate these characters with a distinct style, you define a style run. You can use the `ATSUSetRunStyle` function to define a style run, by associating a style object with a run of text in a text layout object. There is a limit of 64K different styles for each ATSUI text layout object. Each text run must be assigned its own style object, which may or may not differ from other style objects assigned to other text runs in a given text layout object.

You can create a new style object containing only default settings by calling the function `ATSUCreateStyle` (page 1865). To make changes to the default style attributes, you can call the function `ATSUSetAttributes` (page 1950). To set font features and font variations, call the functions `ATSUSetFontFeatures` (page 1952) and `ATSUSetVariations` (page 1967), respectively.

Note that if you call `ATSUSetRunStyle` on a text run that is already associated with a style object, the style set by `ATSUSetRunStyle` overrides the previous style. Additionally, upon completion, `ATSUSetRunStyle` adjusts the lengths of any style runs on either side of the affected style run.

For example, you may currently have a run of text, 40 characters long, that is assigned a single style, styleA. If you call `ATSUSetRunStyle`, you can reassign characters at offset 10–29 to a new style, styleB. If you do so, you would then have three style runs, where there once was one: characters at offset 0–9 (styleA), 10–29 (styleB), and 30–39 (styleA).

After calling `ATSUSetRunStyle`, you can call the function `ATSUDrawText` (page 1877) to display the styled text. When you call `ATSUDrawText`, if you have not previously assigned styles to all the characters you request to be drawn, ATSUI automatically does so. Specifically, ATSUI extends the first style it locates immediately prior (in storage order) to the unstyled characters to include those unassigned characters. If the unstyled characters are at the beginning of the text stream, ATSUI finds the first style run in the stream and extends it backward to the first character.

You should call `ATSUSetRunStyle` whenever you create a new text layout object without any associated styles, as by using the function `ATSUCreateTextLayout` (page 1866). You should also call `ATSUSetRunStyle` to assign a style to a text run in response to a user action, such as when the user selects a run of text and changes the font.

You do not need to call `ATSUSetRunStyle` when you change style attributes or text layout attributes. In such cases, ATSUI automatically updates the layout of the text as appropriate.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeObjects.h`

ATSUSetSoftLineBreak

Sets a soft line break that you specify.

```
OSStatus ATSUSetSoftLineBreak (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iLineBreak
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object for which to set a line break.

iLineBreak
A `UniCharArrayOffset` value specifying the offset from the beginning of the text layout object's text buffer to the line break to set.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUSetSoftLineBreak` function enables you to set a soft line break in a text range. You should typically only call `ATSUSetSoftLineBreak` to set line breaks when you are using your own line-breaking algorithm to calculate these breaks. For optimal performance, you should use `ATSUBatchBreakLines` (page 1844) to both calculate and set soft line breaks in your text.

After calling `ATSUSetSoftLineBreak`, you should call the function `ATSUGetUnjustifiedBounds` (page 1923) to determine whether the characters still fit within the line, which is necessary due to end-of-line effects such as swashes.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

ATSUSetStyleRefCon

Sets application-specific data for a style object.

```
OSStatus ATSUSetStyleRefCon (
   ATSUStyle iStyle,
   URefCon iRefCon
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to set application-specific data.

iRefCon

A 32-bit value containing or referring to application-specific style data.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUSetStyleRefCon` function associates a reference constant (that is, application-specific data) with a style object. If you copy or clear a style object that contains a reference constant, the reference constant is neither copied nor removed. To obtain application-specific data for a style object, call the function `ATSUGetStyleRefCon` (page 1918).

When you dispose of a style object that contains a reference constant, you are responsible for freeing any memory allocated for the reference constant. Calling the function `ATSUDisposeStyle` (page 1876) does not do so.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUSetTabArray

Sets a tab ruler for a text layout object.

```
OSStatus ATSUSetTabArray (
   ATSUTextLayout iTextLayout,
   const ATSUTab iTabs[],
   ItemCount iTabCount
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object for which you want to set a tab ruler.

iTabs[]

An array of the tab values you want applied to the text layout object. This tab ruler is applied to all lines in the text layout object. You can pass `NULL` if `iTabCount` equals `0`. Passing `NULL` effectively deletes any tab ruler that was set previously.

iTabCount

The number of tabs in the given `iTabs` array. If value is `0`, any previously-set tab ruler is cleared from the text layout object.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion
When a tab ruler is set for a text layout object, ATSUI automatically aligns text such that any tabs in the text are laid out to follow the tab ruler's specifications. If you want to use tabs in your text and you also want to use the function `ATSUBatchBreakLines`, then you must set tabs by calling the function `ATSUSetTabArray`.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUSetTextHandleLocation

Associates text with a text layout object. (Deprecated in Mac OS X v10.0. Use `ATSUSetTextPointerLocation` (page 1965) instead. See the Discussion for more details.)

Not recommended.

```
OSStatus ATSUSetTextHandleLocation (
   ATSUTextLayout iTextLayout,
   UniCharArrayHandle iText,
   UniCharArrayOffset iTextOffset,
   UniCharCount iTextLength,
   UniCharCount iTextTotalLength
);
```

Parameters

iTextLayout
An `ATSUTextLayout` value specifying the text layout object with which to associate text.

iText
A handle of type `UniCharArrayHandle`, referring to a text buffer containing UTF-16–encoded text. ATSUI associates this buffer with the text layout object and analyzes the complete text of the buffer when obtaining the layout context for the current text range. Thus, for paragraph-format text, if you specify a buffer containing less than a complete paragraph, some of ATSUI's layout results are not guaranteed to be accurate. For example, with a buffer of less than a full paragraph, ATSUI can neither reliably obtain the context for bidirectional processing nor reliably generate accent attachments and ligature formations for Roman text.

iTextOffset
A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the first character of the range to include in the layout. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`,. To specify the entire text buffer, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iTextLength` parameter.

iTextLength
A `UniCharCount` value specifying the length of the text range. Note that `iTextOffset + iTextLength` must be less than or equal to the value of the *iTextTotalLength* parameter. If you want the range of text to extend to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

iTextTotalLength

A UniCharCount value specifying the length of the entire text buffer. This value should be greater than or equal to the range of text defined by the iTextLength parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should use the function ATSUSetTextPointerLocation (page 1965) instead of the function ATSUSetTextHandleLocation.

For ATSUI to render your text, you must associate the text with both a text layout object and style information. Some functions, such as ATSUCreateTextLayoutWithTextPtr (page 1869), create a text layout object and associate text with it concurrently. However, if you use the function ATSUCreateTextLayout (page 1866) to create a layout object, you must assign text to the text layout object prior to attempting most ATSUI operations.

You can use either of the functions ATSUSetTextHandleLocation or ATSUSetTextPointerLocation (page 1965) to associate text with a text layout object. When you call these functions, you are both assigning a text buffer to a text layout object and specifying the current text subrange within the buffer to include in the layout.

If there is already text associated with a text layout object, calling ATSUSetTextHandleLocation or ATSUSetTextPointerLocation overrides the previously associated text, as well as clearing the object's layout caches. You would typically only call these functions for a text layout object with existing associated text if either (a) both the buffer itself is relocated and a subrange of the buffer's text is deleted or inserted or (b) when associating an entirely different buffer with a text layout object.

Note that, because ATSUI objects retain state, doing superfluous calling can degrade performance. For example, you could call ATSUSetTextHandleLocation rather than ATSUTextInserted (page 1970) when the user simply inserts a subrange of text within a text buffer, but there would be a performance penalty, as all the layout caches are flushed by ATSUSetTextHandleLocation, rather than just the affected ones.

Similarly, you should not call ATSUSetTextHandleLocation, when an entire text buffer associated with a text layout object is relocated, but no other changes have occurred that would affect the buffer's current subrange. Instead, you should call ATSUTextMoved (page 1971), which is a more focused function and therefore more efficient.

After associating text with the text layout object, use ATSUSetRunStyle (page 1959) to associate style information with the text. You can then call the function ATSUDrawText (page 1877) to display the text.

Note that while ATSUSetTextHandleLocation associates text with a text layout object via a handle, ATSUI functions that need to access the text return the handle to its original state upon function completion.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.0.
Not available to 64-bit applications.

Declared In
ATSUnicodeObjects.h

ATSUSetTextLayoutRefCon

Sets application-specific data for a text layout object.

```
OSStatus ATSUSetTextLayoutRefCon (
   ATSUTextLayout iTextLayout,
   URefCon iRefCon
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object for which to set application-specific data.

iRefCon

A 32-bit value containing or referring to application-specific text layout data.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The `ATSUSetTextLayoutRefCon` function associates a reference constant (that is, application-specific data) with a text layout object. You might typically use `ATSUSetTextLayoutRefCon` to track user preferences that can effect layout, for example.

If you copy or clear a text layout object containing a reference constant, the reference constant is not copied or removed. When you dispose of a text layout object that contains a reference constant, you are responsible for freeing any memory allocated for the reference constant. Calling the function `ATSUDisposeTextLayout` (page 1876) does not do so.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeObjects.h`

ATSUSetTextPointerLocation

Associates text with a text layout object or updates previously associated text.

```
OSStatus ATSUSetTextPointerLocation (
   ATSUTextLayout iTextLayout,
   ConstUniCharArrayPtr iText,
   UniCharArrayOffset iTextOffset,
   UniCharCount iTextLength,
   UniCharCount iTextTotalLength
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object for which to set text.

iText

A pointer of type `ConstUniCharArrayPtr`, referring to a text buffer containing UTF-16–encoded text. ATSUI associates this buffer with the text layout object and analyzes the complete text of the buffer when obtaining the layout context for the current text range. Thus, for paragraph-format text, if you specify a buffer containing less than a complete paragraph, some of ATSUI's layout results are not guaranteed to be accurate. For example, with a buffer of less than a full paragraph, ATSUI can neither reliably obtain the context for bidirectional processing nor reliably generate accent attachments and ligature formations.

iTextOffset

A UniCharArrayOffset value specifying the offset from the beginning of the text buffer to the first character of the range to include in the layout. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant kATSUFromTextBeginning. To specify the entire text buffer, pass kATSUFromTextBeginning in this parameter and kATSUToTextEnd in the iTextLength parameter.

iTextLength

A UniCharCount value specifying the length of the text range. Note that iTextOffset + iTextLength must be less than or equal to the value of the *iTextTotalLength* parameter. If you want the range of text to extend to the end of the text buffer, you can pass the constant kATSUToTextEnd.

iTextTotalLength

A UniCharCount value specifying the length of the entire text buffer. This value should be greater than or equal to the range of text defined by the iTextLength parameter.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
For ATSUI to render your text, you must associate the text with both a text layout object and style information. Some functions, such as ATSUCreateTextLayoutWithTextPtr (page 1869), create a text layout object and associate text with it concurrently. However, if you use the function ATSUCreateTextLayout (page 1866) to create a text layout object, you must assign text to the object prior to attempting most ATSUI operations.

You can use the function ATSUSetTextPointerLocation or to associate text with a text layout object. When you call this function, you are both assigning a text buffer to a text layout object and specifying the current text subrange within the buffer to include in the layout.

If there is already text associated with a text layout object, calling ATSUSetTextPointerLocation overrides the previously associated text, as well as clearing the object's layout caches. You would typically only call this function for a text layout object with existing associated text if either (a) both the buffer itself is relocated and a subrange of the buffer's text is deleted or inserted or (b) when associating an entirely different buffer with a text layout object.

Note that, because ATSUI objects retain state, doing superfluous calling can degrade performance. For example, you could call ATSUSetTextPointerLocation rather than ATSUTextInserted (page 1970) when the user simply inserts a subrange of text within a text buffer, but there would be a performance penalty, as all the layout caches are flushed by ATSUSetTextPointerLocation, rather than just the affected ones.

Similarly, you should not call ATSUSetTextPointerLocation, when an entire text buffer associated with a text layout object is relocated, but no other changes have occurred that would affect the buffer's current subrange. Instead, you should call ATSUTextMoved (page 1971), which is a more focused function and therefore more efficient.

After associating text with a text layout object, use ATSUSetRunStyle (page 1959) to associate style information with the text. You can then call the function ATSUDrawText (page 1877) to display the text or a subrange of the text.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeObjects.h

ATSUSetTransientFontMatching

Turns automatic font substitution on or off for a text layout object.

```
OSStatus ATSUSetTransientFontMatching (
   ATSUTextLayout iTextLayout,
   Boolean iTransientFontMatching
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object for which to set automatic font substitution on or off.

iTransientFontMatching

A `Boolean` value indicating whether ATSUI is to perform automatic font substitution for the text layout object. If you pass `true`, ATSUI performs automatic font substitution for the text range associated with the text layout object.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

Calling the `ATSUSetTransientFontMatching` function sets ATSUI's automatic font substitution to on or off for a given text layout object. When automatic font substitution is on, ATSUI scans the text range associated with specified text layout object looking for undrawable characters whenever a layout is performed, for example, when text is measured or drawn. When ATSUI finds a character that cannot be drawn with the currently assigned font, it identifies a valid font for the character and draws the character. ATSUI continues scanning the text range for characters in need of substitute fonts, replacing the font and redrawing the characters as needed. ATSUI stops scanning when it reaches the end of the text range associated with the text layout object.

ATSUI's default behavior for finding a substitute font is to use the first valid font that it finds when sequentially scanning the fonts in the user's system. However, you can alter this behavior by calling the function `ATSUCreateFontFallbacks` (page 1864) and defining your own font fallback settings for the text layout object. If ATSUI cannot find any suitable replacement fonts, it substitutes the missing-character glyph—that is, a glyph representing an empty box—to indicate to the user that a valid font is not installed on their system.

Note that when `ATSUSetTransientFontMatching` performs font substitution, it does not change the font attribute in the associated style object. That is, the font attribute for the style object associated with the redrawn character(s) remains set to the invalid font—not the valid substitute font— just as it was prior to calling `ATSUSetTransientFontMatching`.

If you want ATSUI to identify a substitute font, but you do not want ATSUI to automatically perform the font substitution, you can call the function `ATSUMatchFontsToText` (page 1936).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSUnicodeObjects.h`

ATSUSetVariations

Sets font variation axes and values in a style object.

```
OSStatus ATSUSetVariations (
   ATSUStyle iStyle,
   ItemCount iVariationCount,
   const ATSUFontVariationAxis iAxes[],
   const ATSUFontVariationValue iValue[]
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object for which to set font variation values.

iVariationCount

An `ItemCount` value specifying the number of font variation values to set. This value should correspond to the number of elements in the `iAxes` and `iValue` arrays.

iAxes

A pointer to the initial `ATSUFontVariationAxis` value in an array of font variation axes. Each element in the array must represent a valid variation axis tag that corresponds to a variation value in the *iValue* array. To obtain a valid variation axis tag for a font, you can call the functions `ATSUGetIndFontVariation` (page 1911) or `ATSUGetFontInstance` (page 1900).

iValue

A pointer to the initial `ATSUFontVariationValue` value in an array of font variation values. Each element in the array must contain a value that is valid for the corresponding variation axis in the `iAxes` parameter. You can obtain a font's maximum, minimum, and default values for a given variation axis by calling the function `ATSUGetIndFontVariation` (page 1911). You can obtain the font variation axis values for a font instance by calling `ATSUGetFontInstance` (page 1900).

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

If you supply font variation axes and values to the `ATSUSetVariations` function, you can change the appearance of a style object's font accordingly. You may specify any number of variation axes and values in a style object. Any of the font's variations that you do not set retain their font-defined default values.

You can also use the `ATSUSetVariations` function to supply your own value within any variation axes defined for the font. However, if the font does not support the variation axis you specify, your custom variation has no visual effect.

By calling the function `ATSUGetIndFontVariation` (page 1911), you can obtain a variation axis and its maximum, minimum, and default values for a font.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeFonts.h`

ATSUStyleIsEmpty

Indicates whether a style object contains only default values.

```
OSStatus ATSUStyleIsEmpty (
   ATSUStyle iStyle,
   Boolean *oIsClear
);
```

Parameters

iStyle

An `ATSUStyle` value specifying the style object to examine.

oIsClear

A pointer to a `Boolean` value. On return, the value is set to `true` if the style object contains only default values for style attributes, font features, and font variations. If `false`, the style object contains one or more nondefault values for style attributes, font features, or font variations.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

You can call the `ATSUStyleIsEmpty` function to determine whether a style object contains only default values for style attributes, font features, and font variations. `ATSUStyleIsEmpty` does not consider reference constants in its evaluation.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUTextDeleted

Informs ATSUI of the location and length of a text deletion.

```
OSStatus ATSUTextDeleted (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iDeletedRangeStart,
   UniCharCount iDeletedRangeLength
);
```

Parameters

iTextLayout

An `ATSUTextLayout` value specifying the text layout object containing the deleted text.

iDeletedRangeStart

A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the memory location of the deleted text. To specify a deletion point at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`. To specify that the entire text buffer has been deleted, pass `kATSUFromTextBeginning` in this parameter and `kATSUToTextEnd` in the `iDeletedRangeLength` parameter.

iIDeletedRangeLength

A `UniCharCount` value specifying the length of the deleted text. To specify a deletion length extending to the end of the text buffer, you can pass the constant `kATSUToTextEnd`.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion
When you call the `ATSUTextDeleted` function to inform ATSUI of a text deletion, it shortens the style run(s) containing the deleted text by the amount of the deletion. If a style run corresponds entirely to a range of deleted text, that style run is removed. If the deletion point is between two style runs, the first style run is shortened (or removed).

The `ATSUTextDeleted` function also shortens the total length of the text buffer containing the deleted text by the amount of the deletion. That is, it shifts the memory location of the text following the deleted text by `iDeletedRangeLength`. `ATSUTextDeleted` also removes any soft line breaks that fall within the deleted text and updates affected drawing caches.

The `ATSUTextDeleted` function does not change the actual memory location of the affected text. You are responsible for deleting the corresponding text is from the text buffer. You are also responsible for calling the function `ATSUDisposeStyle` (page 1876) to dispose of the memory associated with any style runs that have been removed.

Note that calling the function `ATSUTextDeleted` automatically removes previously-set soft line breaks if the line breaks are within the range of text that is deleted.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeObjects.h`

ATSUTextInserted

Informs ATSUI of the location and length of a text insertion.

```
OSStatus ATSUTextInserted (
   ATSUTextLayout iTextLayout,
   UniCharArrayOffset iInsertionLocation,
   UniCharCount iInsertionLength
);
```

Parameters
iTextLayout
An `ATSUTextLayout` value specifying the text layout object containing the inserted text.
iInsertionLocation
A `UniCharArrayOffset` value specifying the offset from the beginning of the text buffer to the memory location of the inserted text. To specify an insertion point at the beginning of the text buffer, you can pass the constant `kATSUFromTextBeginning`.
iInsertionLength
A `UniCharCount` value specifying the length of the inserted text.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
When you call the `ATSUTextInserted` function to inform ATSUI of a text insertion, it extends the style run containing the insertion point by the amount of the inserted text. If the insertion point is between two style runs, the first style run is extended to include the new text.

The ATSUTextInserted function also extends the total length of the text buffer containing the inserted text by the amount of the inserted text. That is, it shifts the memory location of the text following the inserted text by iInsertionLength. ATSUTextInserted then updates drawing caches.

Note that the ATSUTextInserted function does not change the actual memory location of the inserted text. You are responsible for placing the inserted text into the text buffer at the appropriate location.

The ATSUTextInserted function does not insert style runs or line breaks; to do so, call the functions ATSUSetRunStyle (page 1959) and ATSUSetSoftLineBreak (page 1961), respectively. Break line operations should be redone after you call ATSUTextInserted.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeObjects.h

ATSUTextMoved

Informs ATSUI of the new memory location of relocated text.

```
OSStatus ATSUTextMoved (
   ATSUTextLayout iTextLayout,
   ConstUniCharArrayPtr iNewLocation
);
```

Parameters

iTextLayout
An ATSUTextLayout value identifying the text layout object associated with the relocated text.

iNewLocation
A ConstUniCharArrayPtr specifying the new memory location of the moved text.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should call the ATSUTextMoved function when a range of text consisting of less than an entire text buffer has been moved. The ATSUTextMoved function informs ATSUI of the new memory location of the text. You are responsible for moving the text. The text buffer should remain otherwise unchanged.

When a range of text consisting of an entire text buffer has been moved, you should:

- Call the function ATSUSetTextPointerLocation (page 1965) to update the text buffer's location.
- Call the function ATSUSetRunStyle (page 1959) to update the corresponding style runs for the text buffer.
- Call the function ATSUDrawText (page 1877) to display the updated text.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeObjects.h

ATSUUnderwriteAttributes

Copies to a destination style object only those nondefault style attribute settings of a source style object that are at default settings in the destination object.

```
OSStatus ATSUUnderwriteAttributes (
   ATSUStyle iSourceStyle,
   ATSUStyle iDestinationStyle
);
```

Parameters

iSourceStyle

An `ATSUStyle` value specifying the style object from which to copy nondefault style attributes.

iDestinationStyle

An `ATSUStyle` value specifying the style object containing style attribute values to be set.

Return Value

A result code. See "ATSUI Result Codes" (page 2068).

Discussion

The `ATSUUnderwriteAttributes` function copies to a destination style object only those nondefault style attribute values of a source style object that are not currently set in a destination style object. Note that the corresponding value in the destination object must not be set in order for a copied value to be applied. All other quantities in the destination style object are left unchanged.

`ATSUUnderwriteAttributes` does not copy the contents of memory referenced by pointers within custom style attributes or within reference constants. You are responsible for ensuring that this memory remains valid until both the source and destination style objects are disposed of.

To create a style object that contains all the contents of another style object, call the function `ATSUCreateAndCopyStyle` (page 1863). To copy all the style attributes (including any default settings) of a style object into an existing style object, call the function `ATSUCopyAttributes` (page 1856). To copy style attributes that are set in the source whether or not they are set in the destination style object, call the function `ATSUOverwriteAttributes` (page 1944).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ATSUnicodeObjects.h`

ATSUUnflattenStyleRunsFromStream

Unflattens previously-flattened ATSUI style run data so that it can be read from disk or accepted (through the pasteboard) from another application.

```
OSStatus ATSUUnflattenStyleRunsFromStream (
   ATSUFlattenedDataStreamFormat iStreamFormat,
   ATSUUnFlattenStyleRunOptions iUnflattenOptions,
   ByteCount iStreamBufferSize,
   const void *iStreamBuffer,
   ItemCount iNumberOfRunInfo,
   ItemCount iNumberOfStyleObjects,
   ATSUStyleRunInfo oRunInfoArray[],
   ATSUStyle oStyleArray[],
   ItemCount *oActualNumberOfRunInfo,
   ItemCount *oActualNumberOfStyleObjects
);
```

Parameters

iStreamFormat

The format of the flattened data. There is only one format supported at this time (`'ustl'`) so you must pass the constant `kATSUDataStreamUnicodeStyledText`.

iUnflattenOptions

The options you want to use to unflatten the data. There are no options supported at this time, so you must pass the constant `kATSUUnflattenOptionsNoOptionsMask`.

iStreamBufferSize

The size of the buffer pointed to by the `iStreamBuffer` parameter. You must pass a value greater than `0`.

iStreamBuffer

A pointer to the buffer that contains the flattened data. The data must be of the format specified by the `iStreamFormat` parameter and must be of size specified by the `iStreamBufferSize` parameter. You cannot pass `NULL`.

iNumberOfRunInfo

The number of style run information structures passed in the `iRunInfoArray` parameter. If you are uncertain of the number of style run information structures, see the Discussion.

iNumberOfStyleObjects

The number of `ATSUStyle` objects in the array passed into the `iStyleArray` parameter. If you are uncertain of the number of `ATSUStyle` objects, see the Discussion.

oRunInfoArray[]

On return, points to an array of style run information structures. Each structure contains a style run length and index into the `oStyleArray` array. If you are uncertain of how much memory to allocate for this array, see the Discussion. You are responsible for disposing of the array when you no longer need it.

oStyleArray[]

On return, a pointer to an array of the unique ATSUI style objects (`ATSUStyle`) obtained from the flattened data. The indices returned in the array `oRunInfoArray` are indices into this array. If you are uncertain of how much memory to allocate for this array, see the Discussion. You are responsible for disposing of the array and the ATSUI style objects in the array when you no longer need the array.

oActualNumberOfRunInfo

On return, points to the actual number of `ATSUStyleRunInfo` structures obtained from the flattened data. The actual number of structures is the number of entries added to the array `oRunInfoArray`. You can pass `NULL` if you to not want to obtain this value.

oActualNumberOfStyleObjects

On return, points to the actual number of unique ATSUI style objects (ATSUStyle) obtained from the flattened data. The actual number is the number of entries added to the oStyleArray array. You can pass NULL if you do no want to obtain this value.

Return Value
A result code. See "ATSUI Result Codes" (page 2068). This function can also return paramErr if you pass invalid values for any of the parameters.

Discussion
The function ATSUUnflattenStyleRunsFromStream extracts the ATSUI style run information from previously-flattened data. The style objects and style run information structures are returned in two separate arrays—the array oStyleArray and the array oRunInfoArray. These arrays are not parallel. Each ATSUStyle object in the oStyleArray is a unique ATSUStyle object. To figure out which ATSUStyle object belongs to which text run, the caller must parse the array of ATSUStyleRunInfo structures. These structures contain the style run lengths and an index into the oStyleArray.

Typically you use the function ATSUUnflattenStyleRunsFromStream by calling it twice, as follows:

1. Provide appropriate values for the iStreamFormat, iUnflattenOptions, and iStreamBuffer parameters. Pass 0 for the iNumberOfRunInfo and iNumberOfStyleObjects parameters, NULL for the oRunInfoArray and oStyleArray, parameters and valid ItemCount references for the oActualNumberOfRunInfo and oActualNumberOfStyleObjects parameters. On return, oActualNumberOfRunInfo and oActualNumberOfStyleObjects point to the sizes needed to allocate these arrays.

2. Allocate appropriately-sized arrays of ATSUStyleRunInfo data structures and ATSUStyle objects. Call the function ATSUUnflattenStyleRunsFromStream a second time, passing the newly allocated arrays in the oRunInfoArray and oStyleArray parameters, with the iNumberOfRunInfo and iNumberOfStyleObjects parameters set to the values you obtained from the first call.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
ATSUnicodeFlattening.h

ATSUUnhighlightText

Renders a previously highlighted range of text in an unhighlighted state.

```
OSStatus ATSUUnhighlightText (
   ATSUTextLayout iTextLayout,
   ATSUTextMeasurement iTextBasePointX,
   ATSUTextMeasurement iTextBasePointY,
   UniCharArrayOffset iHighlightStart,
   UniCharCount iHighlightLength
);
```

Parameters
iTextLayout

An ATSUTextLayout value identifying the text layout object for which to render unhighlighted text.

iTextBasePointX

An ATSUTextMeasurement value specifying the x-coordinate of the origin (in either the current graphics port or in a Quartz graphics context) of the line containing the text range. Pass the constant kATSUUseGrafPortPenLoc, described in "Convenience Constants" (page 2043), to draw relative to the current pen location in the current graphics port.

iTextBasePointY

An ATSUTextMeasurement value specifying the y-coordinate of the origin (in either the current graphics port or in a Quartz graphics context) of the line containing the text range. Pass the constant kATSUUseGrafPortPenLoc, described in "Convenience Constants" (page 2043), to draw relative to the current pen location in the current graphics port.

iHighlightStart

A UniCharArrayOffset value specifying the offset from the beginning of the text buffer to the first character of the text range. If the text range spans multiple lines, you should call ATSUUnhighlightText for each line, passing the offset corresponding to the beginning of the new line to draw with each call. To indicate that the specified text range starts at the beginning of the text buffer, you can pass the constant kATSUFromTextBeginning. To specify the entire text buffer, pass kATSUFromTextBeginning in this parameter and kATSUToTextEnd in the iHighlightLength parameter.

iHighlightLength

A UniCharCount value specifying the length of the text range. To indicate that the text range extends to the end of the text buffer, pass the constant kATSUToTextEnd.

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
The ATSUUnhighlightText function renders a previously highlighted range of text in an unhighlighted state. You should always call ATSUUnhighlightText after calling the function ATSUHighlightText (page 1931), to properly redraw the unhighlighted text and background.

If the inversion method of highlighting was used, when you call ATSUUnhighlightText, it merely undoes the inversion and renders the text.

If the redraw method of highlighting was used, ATSUUnhighlightText turns off the highlighting and restores the desired background. Depending on the complexity of the background, ATSUI restores the background in one of two ways:

- When the background is a single color (such as white), ATSUI can readily unhighlight the background. In such a case, you specify the background color that ATSUI uses by calling the function ATSUSetHighlightingMethod (page 1953) and setting iMethod to kRedrawHighlighting and iUnhighlightData.dataType to kATSUBackgroundColor and providing the background color in iUnhighlightData.unhighlightData. With these settings defined, when you call ATSUUnhighlightText, ATSUI simply calculates the previously highlighted area, repaints it with the specified background color, and then redraws the text.
- When the background is more complex (containing, for example, multiple colors, patterns, or pictures), you must provide a redraw background callback function when you call ATSUSetHighlightingMethod. You do this by setting iUnhighlightData.dataType to kATSUBackgroundCallback and providing a RedrawBackgroundUPP in iUnhighlightData.unhighlightData. When ATSUI calls your callback, you are responsible for redrawing the background of the unhighlighted area. If you choose to also redraw the text, then your callback should return false as a function result. If your callback returns true ATSUI redraws any text that needs to be redrawn. See RedrawBackgroundProcPtr (page 1999) for additional information.

Before calculating the dimensions of the area to unhighlight, `ATSUUnhighlightText` examines the text layout object to ensure that each of the characters in the range is assigned to a style run. If there are gaps between style runs, ATSUI assigns the characters in the gap to the style run that precedes (in storage order) the gap. If there is no style run at the beginning of the text range, ATSUI assigns these characters to the first style run it finds. If there is no style run at the end of the text range, ATSUI assigns the remaining characters to the last style run it finds.

The `ATSUUnhighlightText` function uses the previously set line ascent and descent values to calculate the height of the region to unhighlight. If these values have not been set for the line, `ATSUUnhighlightText` uses the line ascent and descent values set for the text layout object containing the line. If these are not set, it uses the default values.

If you want to remove highlighting from a text range that spans multiple lines, you should call `ATSUUnhighlightText` for each line of text that is being unhighlighted, even if all the lines belong to the same text layout object. You should adjust the `iHighlightStart` parameter to reflect the beginning of each line to be unhighlighted.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ATSUnicodeDrawing.h`

DisposeATSCubicClosePathUPP

Disposes of a universal procedure pointer (UPP) to a cubic close-path callback.

```
void DisposeATSCubicClosePathUPP (
   ATSCubicClosePathUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `ATSCubicClosePathProcPtr` (page 1990) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

DisposeATSCubicCurveToUPP

Disposes of a universal procedure pointer (UPP) to a cubic curve-to callback.

```
void DisposeATSCubicCurveToUPP (
   ATSCubicCurveToUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `ATSCubicCurveToProcPtr` (page 1991) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

DisposeATSCubicLineToUPP

Disposes of a universal procedure pointer (UPP) to a cubic line-to callback.

```
void DisposeATSCubicLineToUPP (
   ATSCubicLineToUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `ATSCubicLineToProcPtr` (page 1992) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

DisposeATSCubicMoveToUPP

Disposes of a universal procedure pointer (UPP) to a cubic move-to callback.

```
void DisposeATSCubicMoveToUPP (
   ATSCubicMoveToUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `ATSCubicMoveToProcPtr` (page 1993) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeGlyphs.h

DisposeATSQuadraticClosePathUPP

Disposes of a universal procedure pointer (UPP) to a quadratic close-path callback.

```
void DisposeATSQuadraticClosePathUPP (
   ATSQuadraticClosePathUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback ATSQuadraticClosePathProcPtr (page 1994) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeGlyphs.h

DisposeATSQuadraticCurveUPP

Disposes of a universal procedure pointer (UPP) to a quadratic curve callback.

```
void DisposeATSQuadraticCurveUPP (
   ATSQuadraticCurveUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback ATSQuadraticCurveProcPtr (page 1995) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeGlyphs.h

DisposeATSQuadraticLineUPP

Disposes of a universal procedure pointer (UPP) to a quadratic line callback.

```
void DisposeATSQuadraticLineUPP (
   ATSQuadraticLineUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `ATSQuadraticLineProcPtr` (page 1996) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

DisposeATSQuadraticNewPathUPP

Disposes of a universal procedure pointer (UPP) to a quadratic new-path callback.

```
void DisposeATSQuadraticNewPathUPP (
   ATSQuadraticNewPathUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `ATSQuadraticNewPathProcPtr` (page 1997) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

DisposeATSUDirectLayoutOperationOverrideUPP

Disposes of a universal procedure pointer (UPP) to a layout operation override callback.

```
void DisposeATSUDirectLayoutOperationOverrideUPP (
   ATSUDirectLayoutOperationOverrideUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
ATSLayoutTypes.h

DisposeRedrawBackgroundUPP

Disposes of a new universal procedure pointer (UPP) to a redraw background callback.

```
void DisposeRedrawBackgroundUPP (
   RedrawBackgroundUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer.

Discussion
See the callback RedrawBackgroundProcPtr (page 1999) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeTypes.h

InvokeATSCubicClosePathUPP

Calls your cubic close-path callback.

```
OSStatus InvokeATSCubicClosePathUPP (
   void *callBackDataPtr,
   ATSCubicClosePathUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function InvokeATSCubicClosePathUPP, as ATSUI calls your cubic close-path callback for you. See the callback ATSCubicClosePathProcPtr (page 1990) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
ATSUnicodeGlyphs.h

InvokeATSCubicCurveToUPP

Calls your cubic curve-to callback.

```
OSStatus InvokeATSCubicCurveToUPP (
   const Float32Point *pt1,
   const Float32Point *pt2,
   const Float32Point *pt3,
   void *callBackDataPtr,
   ATSCubicCurveToUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSCubicCurveToUPP`, as ATSUI calls your cubic curve-to callback for you. See the callback `ATSCubicCurveToProcPtr` (page 1991) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

InvokeATSCubicLineToUPP

Calls your cubic line-to callback.

```
OSStatus InvokeATSCubicLineToUPP (
   const Float32Point *pt,
   void *callBackDataPtr,
   ATSCubicLineToUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSCubicLineToUPP`, as ATSUI calls your cubic line-to callback for you. See the callback `ATSCubicLineToProcPtr` (page 1992) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

InvokeATSCubicMoveToUPP

Calls your cubic move-to callback.

```
OSStatus InvokeATSCubicMoveToUPP (
   const Float32Point *pt,
   void *callBackDataPtr,
   ATSCubicMoveToUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSCubicMoveToUPP`, as ATSUI calls your cubic move-to callback for you. See the callback `ATSCubicMoveToProcPtr` (page 1993) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

InvokeATSQuadraticClosePathUPP

Calls your quadratic close-path callback.

```
OSStatus InvokeATSQuadraticClosePathUPP (
   void *callBackDataPtr,
   ATSQuadraticClosePathUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSQuadraticClosePathUPP`, as ATSUI calls your quadratic close-path callback for you. See the callback `ATSQuadraticClosePathProcPtr` (page 1994) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

InvokeATSQuadraticCurveUPP

Calls your quadratic curve callback.

```
OSStatus InvokeATSQuadraticCurveUPP (
   const Float32Point *pt1,
   const Float32Point *controlPt,
   const Float32Point *pt2,
   void *callBackDataPtr,
   ATSQuadraticCurveUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSQuadraticCurveUPP`, as ATSUI calls your quadratic curve callback for you. See the callback `ATSQuadraticCurveProcPtr` (page 1995) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

InvokeATSQuadraticLineUPP

Calls your quadratic line callback.

```
OSStatus InvokeATSQuadraticLineUPP (
   const Float32Point *pt1,
   const Float32Point *pt2,
   void *callBackDataPtr,
   ATSQuadraticLineUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSQuadraticLineUPP`, as ATSUI calls your quadratic line callback for you. See the callback `ATSQuadraticLineProcPtr` (page 1996) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

InvokeATSQuadraticNewPathUPP

Calls your quadratic new-path callback.

```
OSStatus InvokeATSQuadraticNewPathUPP (
   void *callBackDataPtr,
   ATSQuadraticNewPathUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSQuadraticNewPathUPP`, as ATSUI calls your quadratic new-path callback for you. See the callback `ATSQuadraticNewPathProcPtr` (page 1997) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

InvokeATSUDirectLayoutOperationOverrideUPP

Calls your layout operation override callback.

```
OSStatus InvokeATSUDirectLayoutOperationOverrideUPP (
   ATSULayoutOperationSelector iCurrentOperation,
   ATSULineRef iLineRef,
   URefCon iRefCon,
   void *iOperationCallbackParameterPtr,
   ATSULayoutOperationCallbackStatus *oCallbackStatus,
   ATSUDirectLayoutOperationOverrideUPP userUPP
);
```

Return Value
A result code. See "ATSUI Result Codes" (page 2068).

Discussion
You should not need to use the function `InvokeATSUDirectLayoutOperationOverrideUPP`, as ATSUI calls your layout operation override callback for your. See the callback `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSLayoutTypes.h`

InvokeRedrawBackgroundUPP

Invokes your redraw background callback.

```
Boolean InvokeRedrawBackgroundUPP (
   ATSUTextLayout iLayout,
   UniCharArrayOffset iTextOffset,
   UniCharCount iTextLength,
   ATSTrapezoid iUnhighlightArea[],
   ItemCount iTrapezoidCount,
   RedrawBackgroundUPP userUPP
);
```

Return Value
A `Boolean` value that indicates whether or not the callback was invoked successfully .

Discussion
You should not need to use the function `InvokeRedrawBackgroundUPP`, as ATSUI calls your redraw background callback for you. See the callback `RedrawBackgroundProcPtr` (page 1999) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

NewATSCubicClosePathUPP

Creates a new universal procedure pointer (UPP) to a cubic close-path callback.

```
ATSCubicClosePathUPP NewATSCubicClosePathUPP (
   ATSCubicClosePathProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your cubic close-path callback.

Return Value
On return, a UPP to the cubic close-path callback.

Discussion
See the callback `ATSCubicClosePathProcPtr` (page 1990) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSCubicCurveToUPP

Creates a new universal procedure pointer (UPP) to a cubic curve-to callback.

```
ATSCubicCurveToUPP NewATSCubicCurveToUPP (
   ATSCubicCurveToProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your cubic curve-to callback.

Return Value
On return, a UPP to the cubic curve-to callback.

Discussion
See the callback `ATSCubicCurveToProcPtr` (page 1991) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSCubicLineToUPP

Creates a new universal procedure pointer (UPP) to a cubic line-to callback.

```
ATSCubicLineToUPP NewATSCubicLineToUPP (
   ATSCubicLineToProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your cubic line-to callback.

Return Value
On return, a UPP to the cubic line-to callback.

Discussion
See the callback `ATSCubicLineToProcPtr` (page 1992) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSCubicMoveToUPP

Creates a new universal procedure pointer (UPP) to a cubic move-to callback.

```
ATSCubicMoveToUPP NewATSCubicMoveToUPP (
   ATSCubicMoveToProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your cubic move-to callback.

Return Value
On return, a UPP to the cubic move-to callback.

Discussion
See the callback `ATSCubicMoveToProcPtr` (page 1993) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSQuadraticClosePathUPP

Creates a new universal procedure pointer (UPP) to a quadratic close-path callback.

```
ATSQuadraticClosePathUPP NewATSQuadraticClosePathUPP (
   ATSQuadraticClosePathProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your quadratic close-path callback.

Return Value
On return, a UPP to the quadratic close-path callback.

Discussion
See the callback `ATSQuadraticClosePathProcPtr` (page 1994) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSQuadraticCurveUPP

Creates a new universal procedure pointer (UPP) to a quadratic curve callback.

```
ATSQuadraticCurveUPP NewATSQuadraticCurveUPP (
   ATSQuadraticCurveProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your quadratic curve callback.

Return Value
On return, a UPP to the quadratic curve callback.

Discussion
See the callback `ATSQuadraticCurveProcPtr` (page 1995) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSQuadraticLineUPP

Creates a new universal procedure pointer (UPP) to a quadratic line callback.

```
ATSQuadraticLineUPP NewATSQuadraticLineUPP (
   ATSQuadraticLineProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your quadratic line callback.

Return Value
On return, a UPP to the quadratic line callback.

Discussion
See the callback `ATSQuadraticLineProcPtr` (page 1996) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSQuadraticNewPathUPP

Creates a new universal procedure pointer (UPP) to a quadratic new-path callback.

```
ATSQuadraticNewPathUPP NewATSQuadraticNewPathUPP (
   ATSQuadraticNewPathProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your quadratic new-path callback.

Return Value
On return, a UPP to the quadratic new-path callback.

Discussion
See the callback `ATSQuadraticNewPathProcPtr` (page 1997) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

NewATSUDirectLayoutOperationOverrideUPP

Creates a new universal procedure pointer (UPP) to a layout operation override callback.

```
ATSUDirectLayoutOperationOverrideUPP NewATSUDirectLayoutOperationOverrideUPP (
   ATSUDirectLayoutOperationOverrideProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your layout operation override callback.

Return Value

On return, a UPP to the layout operation override callback.

Discussion

See the callback `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998) for more information.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSLayoutTypes.h`

NewRedrawBackgroundUPP

Creates a new universal procedure pointer (UPP) to a redraw background callback.

```
RedrawBackgroundUPP NewRedrawBackgroundUPP (
   RedrawBackgroundProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your redraw background callback.

Return Value

On return, a UPP to the redraw background callback.

Discussion

See the callback `RedrawBackgroundProcPtr` (page 1999) for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSUnicodeTypes.h`

Callbacks

ATSCubicClosePathProcPtr

Defines a pointer to a cubic close-path callback for drawing glyphs that overrides ATSUI's cubic close-path operation for drawing glyphs.

```
typedef OSStatus(* ATSCubicClosePathProcPtr)
(
    void *callBackDataPtr
);
```

If you name your function `MyATSCubicClosePathCallback`, you would declare it like this:

```
OSStatus MyATSCubicClosePathCallback (
    void *callBackDataPtr
);
```

Parameters

`callBackDataPtr`

A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetCurvePaths` (page 1926). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value

A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetCubicPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion

You supply a pointer to your customized cubic close-path callback as a parameter to the function `ATSUGlyphGetCubicPaths` (page 1925).

To provide a pointer to your cubic close-path callback, you create a universal procedure pointer (UPP) of type `ATSCubicClosePathUPP`, using the function `NewATSCubicClosePathUPP` (page 1985). You can do so with code similar to the following:

```
ATSCubicClosePathUPP MyCubicClosePathUPP;
MyCubicClosePathUPP = NewATSCubicClosePathUPP (&MyATSCubicClosePathCallback);
```

When you no longer need to use your cubic close-path callback, you should use the function `DisposeATSCubicClosePathUPP` (page 1976) to dispose of the universal procedure pointer associated with the callback.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSUnicodeGlyphs.h`

ATSCubicCurveToProcPtr

Defines a pointer to a cubic curve-to callback for drawing glyphs that overrides ATSUI's cubic curve-to operation for drawing glyphs.

```
typedef OSStatus(* ATSCubicCurveToProcPtr)
(
    const Float32Point *pt1,
    const Float32Point *pt2,
    const Float32Point *pt3,
    void *callBackDataPtr
);
```

If you name your function `MyATSCubicCurveToCallback`, you would declare it like this:

```
OSStatus MyATSCubicCurveToCallback (
    const Float32Point *pt1,
    const Float32Point *pt2,
    const Float32Point *pt3,
    void *callBackDataPtr
);
```

Parameters

pt1

A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the first off-curve point for this segment of the glyph.

pt2

A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the second off-curve point for this segment of the glyph.

pt3

A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the end of the curve (an on-curve point) for this segment of the glyph.

callBackDataPtr

A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetCurvePaths` (page 1926). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value

A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetCubicPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion

You supply a pointer to your customized cubic curve-to function as a parameter to the function `ATSUGlyphGetCubicPaths` (page 1925).

To provide a pointer to your cubic curve-to callback, you create a universal procedure pointer (UPP) of type `ATSCubicCurveToUPP`, using the function `NewATSCubicCurveToUPP` (page 1985). You can do so with code similar to the following:

```
ATSCubicCurveToUPP MyCubicCurveToUPP;
MyCubicCurveToUPP = NewATSCubicCurveToUPP (&MyATSCubicCurveToCallback);
```

When you no longer need to use your cubic curve-to callback, you should use the function `DisposeATSCubicCurveToUPP` (page 1976) to dispose of the universal procedure pointer associated with the callback.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSCubicLineToProcPtr

Defines a pointer to a cubic line-to callback for drawing glyphs that overrides ATSUI's cubic line-to operation for drawing glyphs.

```
typedef OSStatus(* ATSCubicLineToProcPtr)
(
    const Float32Point *pt,
    void *callBackDataPtr
);
```

If you name your function `MyATSCubicLineToCallback`, you would declare it like this:

```
OSStatus MyATSCubicLineToCallback (
    const Float32Point *pt,
    void *callBackDataPtr
);
```

Parameters

pt

A `Float32Point` data structure that contains the x and y coordinates for the relative point to which the pen should draw a line.

callBackDataPtr

A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetCurvePaths` (page 1926). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value
A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetCubicPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion
You supply a pointer to your customized cubic line-to callback as a parameter to the function `ATSUGlyphGetCubicPaths` (page 1925).

To provide a pointer to your cubic line-to callback, you create a universal procedure pointer (UPP) of type `ATSCubicLineToUPP`, using the function `NewATSCubicLineToUPP` (page 1986). You can do so with code similar to the following:

```
ATSCubicLineToUPP MyCubicLineToUPP;
MyCubicLineToUPP = NewATSCubicLineToUPP (&MyATSCubicLineToCallback);
```

When you no longer need to use your cubic line-to callback, you should use the function `DisposeATSCubicLineToUPP` (page 1977) to dispose of the universal procedure pointer associated with the callback.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSCubicMoveToProcPtr

Defines a pointer to a cubic move-to function for drawing glyphs that overrides ATSUI's cubic move-to operation for drawing glyphs.

```
typedef OSStatus(* ATSCubicMoveToProcPtr)
(
    const Float32Point *pt,
    void *callBackDataPtr
);
```

If you name your function `MyATSCubicMoveToCallback`, you would declare it like this:

```
OSStatus MyATSCubicMoveToCallback (
    const Float32Point *pt,
    void *callBackDataPtr
);
```

Parameters

pt

A `Float32Point` data structure that contains the x and y coordinates for the relative point to which the pen should move before it begins drawing this segment of the glyph.

callBackDataPtr

A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetCurvePaths` (page 1926). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value
A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetCubicPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion
You supply a pointer to your customized cubic move-to callback as a parameter to the function `ATSUGlyphGetCubicPaths` (page 1925).

To provide a pointer to your cubic move-to callback, you create a universal procedure pointer (UPP) of type `ATSCubicMoveToUPP`, using the function `NewATSCubicMoveToUPP` (page 1986). You can do so with code similar to the following:

```
ATSCubicMoveToUPP MyCubicMoveToUPP;
MyCubicMoveToUPP = ATSCubicMoveToUPP (&MyATSCubicMoveToCallback);
```

When you no longer need to use your cubic move-to callback, you should use the function `DisposeATSCubicMoveToUPP` (page 1977) to dispose of the universal procedure pointer associated with the callback.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticClosePathProcPtr

Defines a pointer to a quadratic close-path callback for drawing glyphs that overrides ATSUI's quadratic close-path operation for drawing glyphs.

```
typedef OSStatus(* ATSQuadraticClosePathProcPtr)
(
    void *callBackDataPtr
);
```

If you name your function `MyATSQuadraticClosePathCallback`, you would declare it like this:

```
OSStatus MyATSQuadraticClosePathCallback
(
    void *callBackDataPtr
);
```

Parameters

`callBackDataPtr`

A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetQuadraticPaths` (page 1928). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value
A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetQuadraticPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion
You supply a pointer to your customized quadratic close-path callback as a parameter to the function `ATSUGlyphGetQuadraticPaths` (page 1928).

To provide a pointer to your quadratic close-path callback, you create a universal procedure pointer (UPP) of type `ATSQuadraticClosePathUPP`, using the function `NewATSQuadraticClosePathUPP` (page 1987). You can do so with code similar to the following:

```
ATSQuadraticClosePathUPP MyQuadraticClosePathUPP;
MyQuadraticClosePathUPP = NewATSQuadraticClosePathUPP
(&MyATSQuadraticClosePathCallback);
```

When you no longer need to use your quadratic close-path callback, you should use the function `DisposeATSQuadraticClosePathUPP` (page 1978) to dispose of the universal procedure pointer associated with the callback.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticCurveProcPtr

Defines a pointer to a quadratic curve callback for drawing glyphs that overrides ATSUI's quadratic curve operation for drawing glyphs.

```
typedef OSStatus(* ATSQuadraticCurveProcPtr)
(
    const Float32Point *pt1,
    const Float32Point *controlPt,
    const Float32Point *pt2,
    void *callBackDataPtr
);
```

If you name your function `MyATSQuadraticCurveCallback`, you would declare it like this:

```
OSStatus MyATSQuadraticCurveCallback (
    const Float32Point *pt1,
    const Float32Point *controlPt,
    const Float32Point *pt2,
    void *callBackDataPtr
);
```

Parameters

pt1

A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the start of the curve (an on-curve point) for this segment of the glyph.

controlPt

A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the control point (an off-curve point) for this segment of the glyph.

pt2

A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the end of the curve (an on-curve point) for this segment of the glyph.

callBackDataPtr

A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetQuadraticPaths` (page 1928). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value
A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetQuadraticPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion
You supply a pointer to your customized quadratic curve callback as a parameter to the function `ATSUGlyphGetQuadraticPaths` (page 1928).

To provide a pointer to your quadratic curve callback, you create a universal procedure pointer (UPP) of type `ATSQuadraticCurveUPP`, using the function `NewATSQuadraticCurveUPP` (page 1987). You can do so with code similar to the following:

```
ATSQuadraticCurveUPP MyQuadraticCurveUPP;
MyQuadraticCurveUPP = NewATSQuadraticCurveUPP (&MyATSQuadraticCurveCallback);
```

When you no longer need to use your quadratic curve callback, you should use the function `DisposeATSQuadraticCurveUPP` (page 1978) to dispose of the universal procedure pointer associated with the callback.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticLineProcPtr

Defines a pointer to a quadratic line callback for drawing glyphs that overrides ATSUI's quadratic line operation for drawing glyphs.

```
typedef OSStatus(* ATSQuadraticLineProcPtr)
(
    const Float32Point *pt1,
    const Float32Point *pt2,
    void *callBackDataPtr
);
```

If you name your function `MyATSQuadraticLineCallback`, you would declare it like this:

```
OSStatus MyATSQuadraticLineCallback (
    const Float32Point *pt1,
    const Float32Point *pt2,
    void *callBackDataPtr
);
```

Parameters

pt1
A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the start of the line for this segment of the glyph.

pt2
A `Float32Point` data structure that contains the x and y coordinates for the relative point that defines the end of the line for this segment of the glyph.

callBackDataPtr
A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetQuadraticPaths` (page 1928). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value
A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetQuadraticPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion
You supply a pointer to your customized quadratic line callback as a parameter to the function `ATSUGlyphGetQuadraticPaths` (page 1928).

To provide a pointer to your quadratic line callback, you create a universal procedure pointer (UPP) of type `ATSQuadraticLineUPP`, using the function `NewATSQuadraticLineUPP` (page 1988). You can do so with code similar to the following:

```
ATSQuadraticLineUPP MyQuadraticLineUPP;
MyQuadraticLineUPP = NewATSQuadraticLineUPP (&MyATSQuadraticLineCallback);
```

When you no longer need to use your quadratic line callback, you should use the function `DisposeATSQuadraticLineUPP` (page 1978) to dispose of the universal procedure pointer associated with the callback.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticNewPathProcPtr

Defines a pointer to a quadratic new-path callback for drawing glyphs that overrides ATSUI's quadratic new-path operation for drawing glyphs.

```
typedef OSStatus(* ATSQuadraticNewPathProcPtr)
(
    void *callBackDataPtr
);
```

If you name your function `MyATSQuadraticNewPathCallback`, you would declare it like this:

```
OSStatus MyATSQuadraticNewPathCallback
(
    void *callBackDataPtr
);
```

Parameters

`callBackDataPtr`

A pointer to any data your callback function needs. You pass this pointer to the function `ATSUGlyphGetQuadraticPaths` (page 1928). Then, ATSUI passes the pointer through to your callback function when your callback function is invoked.

Return Value
A value that indicates the status of your callback function. When a callback function returns any value other than `0`, the `ATSGlyphGetQuadraticPaths` function stops parsing the path outline and returns the result `kATSOutlineParseAbortedErr`.

Discussion
You supply a pointer to your customized quadratic new-path callback as a parameter to the function `ATSUGlyphGetQuadraticPaths` (page 1928).

To provide a pointer to your quadratic new-path callback, you create a universal procedure pointer (UPP) of type `ATSQuadraticNewPathUPP`, using the function `NewATSQuadraticNewPathUPP` (page 1988). You can do so with code similar to the following:

```
ATSQuadraticNewPathUPP MyQuadraticNewPathUPP;
MyQuadraticNewPathUPP = NewATSQuadraticNewPathUPP
(&MyATSQuadraticNewPathCallback);
```

When you no longer need to use your quadratic new-path callback, you should use the function `DisposeATSQuadraticNewPathUPP` (page 1979) to dispose of the universal procedure pointer associated with the callback.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSUDirectLayoutOperationOverrideProcPtr

Defines a pointer to a layout operation callback that overrides an ATSUI layout operation.

```
typedef CALLBACK_API_C (OSStatus, ATSUDirectLayoutOperationOverrideProcPtr
)
    ATSULayoutOperationSelector iCurrentOperation,
    ATSULineRef iLineRef,
    UInt32 iRefCon,
    void *iOperationCallbackParameterPtr,
    ATSULayoutOperationCallbackStatus *oCallbackStatus
);
```

If you name your function `MyLayoutOperationOverrideCallback`, you would declare it like this:

```
OSStatus MyLayoutOperationOverrideCallback
(
    ATSULayoutOperationSelector iCurrentOperation,
    ATSULineRef iLineRef,
    UInt32 iRefCon,
    void *iOperationCallbackParameterPtr,
    ATSULayoutOperationCallbackStatus *oCallbackStatus
);
```

Parameters

iCurrentOperation
The operation that triggered the callback. This value is passed to your callback by ATSUI. If you write a callback that handles more than one layout operation, you can use this value to determine which operation you should handle.

iLineRef
An `ATSULineRef` value that specifies the line of text on which your callback will operation. Your callback gets called for each line of text associated with the text layout object on which you installed the callback.

iRefCon

An unsigned 32-bit integer. This is an optional value. You can use this value to specify any data your application needs, such as user preference data.

iOperationCallbackParameterPtr

A pointer. This is currently unused and should be set to `NULL`.

oCallbackStatus

A layout callback status value. On output, you must supply a status value to indicate to ATSUI whether or not your callback handled the operation. See "Layout Callback Status Values" (page 2055) for a list of the constants you can supply.

Discussion

ATSUI calls your layout operation override function each time the layout operation you specify is invoked. You associate a universal procedure pointer with a text layout object by treating the callback as a layout attribute. That is, you set up a triple (tag, size, value) to specify the layout operation your callback handles, then you call the function `ATSUSetLayoutControls` (page 1955) to associate the triple with the text layout object whose layout operation you want to override. The attribute tag you specify is `kATSULayoutOperationOverrideTag`. The attribute value you specify is an `ATSULayoutOperationOverrideSpecifier` structure that contains a selector for a layout operation and a pointer to your callback function.

To provide a pointer to your layout operation override callback, you create a universal procedure pointer (UPP) of type `ATSUDirectLayoutOperationOverrideUPP`, using the function `NewATSUDirectLayoutOperationOverrideUPP` (page 1989). You can do so with code similar to the following:

```
ATSUDirectLayoutOperationOverrideUPP MyLayoutOperationOverrideUPP;
MyLayoutOperationOverrideUPP = NewATSUDirectLayoutOperationOverrideUPP
                                (&MyLayoutOperationOverrideCallback);
```

When your layout operation is completed, you should use the function `DisposeATSUDirectLayoutOperationOverrideUPP` (page 1979) to dispose of the universal procedure pointer associated with your layout operation override function. However, if you plan to use the same layout operation override function in subsequent layout operations, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

You are limited to the ATSUI functions you can call from within your callback. You can call only those functions that have do not trigger ATSUI to perform the layout operation again. Otherwise, you run the risk of causing infinite recursion. Most functions that use "create", "get", or "copy" semantics are safe to use within your callback. If you call one of the restricted functions, the function returns immediately with the error `kATSUInvalidCallInsideCallbackErr`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSLayoutTypes.h`

RedrawBackgroundProcPtr

Defines a pointer to a redraw-background callback that overrides ATSUI's highlighting method for drawing backgrounds.

```
typedef Boolean (* RedrawBackgroundProcPtr)
(
    ATSUTextLayout iLayout,
    UniCharArrayOffset iTextOffset,
    UniCharCount iTextLength,
    ATSTrapezoid *iUnhighlightArea,
    ItemCount iTrapezoidCount
);
```

If you name your function `MyRedrawBackgroundCallback`, you would declare it like this:

```
Boolean MyRedrawBackgroundCallback (
    ATSUTextLayout iLayout,
    UniCharArrayOffset iTextOffset,
    UniCharCount iTextLength,
    ATSTrapezoid *iUnhighlightArea,
    ItemCount iTrapezoidCount
);
```

Parameters

iLayout

An `ATSUTextLayout` value that specifies the text layout object on which your callback will operate.

iTextOffset

The offset of the text to be highlighted.

iTextLength

The length of the text to be highlighted.

iUnhighlightArea

An array of `ATSTrapezoid` data structures that describe the boundaries of the highlight area. The boundary values in this array are always specified in QuickDraw coordinates.

iTrapezoidCount

The number of `ATSTrapezoid` data structures in the `iUnhighlightArea` array.

Return Value

A `Boolean` value that indicates whether ATSUI should redraw the text. If your function redraws the text, your callback should return `false`, otherwise you callback should return `true` to have ATSUI redraw any text that needs to be redrawn.

Discussion

ATSUI calls your customized redraw-background callback when it needs to redraw complex backgrounds (and optionally the text as well). For ATSUI to use your callback, you must first call the `ATSUSetHighlightingMethod` (page 1953) function with the `iMethod` parameter set to `kRedrawHighlighting`. You must also pass an `ATSUUnhighlightData` data structure as a parameter to the `ATSUSetHighlightingMethod` function. This structure should contain a pointer to your redraw background callback.

To provide a pointer to your redraw background callback, you create a universal procedure pointer (UPP) of type `RedrawBackgroundUPP`, using the function `NewRedrawBackgroundUPP` (page 1989). You can do so with code similar to the following:

```
RedrawBackgroundUPP gMyRedrawBackgroundUPP;
gMyRedrawBackgroundUPP = NewRedrawBackgroundUPP
                            (&MyRedrawBackgroundCallback);
```

For ATSUI to invoke your callback function, you must also pass the `RedrawBackgroundUPP` in the `unhighlightData.backgroundUPP` field of the `iUnhighlightData` parameter for the function `ATSUSetHighlightingMethod`. When finished, you must call the function `DisposeRedrawBackgroundUPP` (page 1980) to dispose of the `RedrawBackgroundUPP`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

Data Types

Core Data Types

ATSUAttributeInfo

Contains an attribute tag and the size of the attribute.

```
struct ATSUAttributeInfo {
    ATSUAttributeTag fTag;
    ByteCount fValueSize;
};
```

Fields

`fTag`

Identifies a particular style run or text attribute value. For a description of the Apple-defined style run and text layout attribute tag constants, see "Attribute Tags" (page 2030).

`fValueSize`

The size (in bytes) of the style run or text layout attribute value.

Discussion
Several ATSUI functions pass back an array of structures of this type. The function `ATSUGetAllAttributes` (page 1886) passes back an array of `ATSUAttributeInfo` structures to represent the data sizes of all previously set style run attribute values and the corresponding style run attribute tags that identify those style run attribute values. The function `ATSUGetAllLayoutControls` (page 1890) passes back an array of `ATSUAttributeInfo` structures to represent the data sizes of all previously set text layout attribute values for an entire text layout object and the corresponding text layout attribute tags that identify those text layout attribute values. The function `ATSUGetAllLineControls` (page 1891) passes back an array of `ATSUAttributeInfo` structures to represent the data sizes of all previously set text layout attribute values for a single line in a text layout object and the corresponding text layout attribute tags that identify those text layout attribute values.

ATSLayoutRecord

Contains basic layout information for a single glyph.

```
struct ATSLayoutRecord {
    ATSGlyphRef         glyphID;
    ATSGlyphInfoFlags   flags;
    ByteCount           originalOffset;
    Fixed               realPos;
};
typedef struct ATSLayoutRecord ATSLayoutRecord;
```

Fields

`glyphID`

A reference to a glyph ID.

`flags`

A flag that specifies the glyph's properties. See "Glyph Property Flags" (page 2051) for the constants you can use.

`originalOffset`

The byte offset of the character with which this glyph is associated.

`realPos`

A `Fixed` value that specifies the real position of the glyph. This is the x-coordinate of the glyph.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSLayoutTypes.h`

ATSUStyleSettingRef

A reference to an opaque style setting object.

```
typedef struct LLCStyleInfo*         ATSUStyleSettingRef;
```

Discussion
You can obtain a style setting reference by calling the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) with the selector set to `kATSUDirectDataStyleSettingATSUStyleSettingRefArray`. You can move a style setting reference from one text layout object to another by calling the function `ATSUDirectAddStyleSettingRef` (page 1871).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeDirectAccess.h`

ATSUAttributeValuePtr

Represents a pointer to a style run or text layout attribute value of unknown size.

```
typedef void* ATSUAttributeValuePtr;
```

Discussion
Each attribute value pointed to by `ATSUAttributeValuePtr` is identified by an attribute tag and the size (in bytes) of the attribute value.

You pass the `ATSUAttributeValuePtr` type to functions that set or clear attribute values in style and text layout objects. The `ATSUAttributeValuePtr` type is passed back by functions that query style and text layout objects for their attribute values. You must dereference this pointer and cast it to the appropriate data type to obtain the actual attribute value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ConstATSUAttributeValuePtr

A pointer to a constant attribute value pointer (`ATSUAttributeValuePtr`).

```
typedef const void* ConstATSUAttributeValuePtr;
```

Discussion
An `ATSUAttributeValuePtr` data type provides generic access to storage of attribute values which vary in size.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSURGBAlphaColor

Contains color information that includes alpha channel information.

```
struct ATSURGBAlphaColor {
    float               red;
    float               green;
    float               blue;
    float               alpha;
};
typedef struct ATSURGBAlphaColor        ATSURGBAlphaColor;
```

Fields

`red`
A value that specifies the red component of the background color.

`green`
A value that specifies the green component of the background color.

`blue`
A value that specifies the blue component of the background color.

`alpha`
A value that specifies thee alpha channel component of the background color.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUBackgroundColor

Redefines the `ATSUBackgroundColor` data type to be an `ATSURGBAlphaColor` data type.

```
typedef ATSURGBAlphaColor ATSUBackgroundColor;
```

Discussion
Prior to Mac OS X version 10.2, the `ATSUBackgroundColor` data type did not include an alpha channel.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUBackgroundData

A union that contains a background color or a universal procedure pointer to a callback that redraws the background.

```
union ATSUBackgroundData {
    ATSUBackgroundColor backgroundColor;
    RedrawBackgroundUPP backgroundUPP;
};
```

Fields
`backgroundColor`
A structure that specifies the background color.

`backgroundUPP`
A universal procedure pointer to a callback function for redrawing complex backgrounds. See `RedrawBackgroundUPP` (page 2030) for more information.

ATSUCaret

Contains the coordinates needed to draw a caret.

```
struct ATSUCaret {
    Fixed fX;
    Fixed fY;
    Fixed fDeltaX;
    Fixed fDeltaY;
};
```

Fields

`fX`

Represents the x-coordinate of the caret's starting pen position relative to the position of the origin of the line in the current graphics port in which the hit occurred.

`fY`

Represents the y-coordinate of the caret's starting pen position relative to the position of the origin of the line in the current graphics port in which the hit occurred.

`fDeltaX`

Represents the x-coordinate of the caret's ending pen position relative to the position of the origin of the line in the current graphics port in which the hit occurred. This position takes into account line rotation. You do not have to rotate it yourself.

`fDeltaY`

Represents the y-coordinate of the caret's ending pen position relative to the position of the origin of the line in the current graphics port in which the hit occurred. This position takes into account line rotation. You do not have to rotate it yourself.

Discussion

The function `ATSUOffsetToPosition` (page 1942) passes back two structures of type `ATSUCaret` to represent the caret position relative to the origin of the line in the current graphics port, corresponding to a specified edge offset. If the edge offset is at a line boundary, the structure passed back in `oMainCaret` contains the starting and ending pen locations of the high caret, while `oSecondCaret` contains the low caret. If the offset is not at a line boundary, both parameters contain the same structure. This structure contains the starting and ending pen locations of the main caret.

You can use the information in this structure to draw a caret by calling the `MoveTo` and `LineTo` functions. For example.

```
MoveTo (fX, fY);
LineTo (fDeltaX, fDeltaY);
```

ATSUFontFeatureType

Represents the attributes of a particular font feature.

```
typedef UInt16 ATSUFontFeatureType;
```

Discussion

Font features are typographic and layout capabilities that you can select or deselect and which control many aspects of glyph selection, ordering, and positioning. Font features include fundamental controls such as whether your text is drawn with contextual forms, as well as details of appearance such as whether you want alternate forms of glyphs to be used at the beginning of a word. To a large extent, how text looks when it is laid out is a function of the number and kinds of font features you choose.

Font vendors create tables that implement the specific set of features which are included in a font by the font designer. Note that only a few feature types and selectors may be available with a given font. If you select features that are not available in a font, you won't see a change in the glyph's appearance. To determine the available features of a font, you can call the functions `ATSUGetFontFeatureTypes` (page 1898) and `ATSUGetFontFeatureSelectors` (page 1897).

For a complete discussion of font features, the selectors you use to access them, and illustrations of the features, see *Inside Mac OS X: Rendering Unicode Text With ATSUI*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUFontFeatureSelector

Represents the state (on or off) of a particular feature type.

```
typedef UInt16 ATSUFontFeatureSelector;
```

Discussion
You pass the `ATSUFontFeatureSelector` type to functions that set or clear font feature selectors in a style run. The `ATSUFontFeatureSelector` type is passed back by functions that obtain font feature selectors in a style run. For a complete discussion of font feature selectors, see *Inside Mac OS X: Rendering Unicode Text With ATSUI*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUFontVariationAxis

Represents a stylistic attribute and the range of values that the font can use to express this attribute.

```
typedef FourCharCode ATSUFontVariationAxis;
```

Discussion
Font variations allow your application to produce a range of type styles algorithmically. You can obtain a a variation axis and its maximum, minimum, and default values for a font by calling the function `ATSUGetIndFontVariation` (page 1911). For a complete discussion of font variations, see *Inside Mac OS X: Rendering Unicode Text With ATSUI*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUFontVariationValue

Represents the range of values that the font can use for a particular font variation.

```
typedef Fixed ATSUFontVariationValue;
```

Discussion
You pass the `ATSUFontVariationValue` type to functions that set and clear font variations in a style run. The `ATSUFontVariationValue` type is passed back by functions that query a style run for font variations.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUFontFallbacks

An opaque structure that contains a font fallback list and font fallback cache information.

```
typedef struct OpaqueATSUFontFallbacks *ATSUFontFallbacks;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUFontID

Represents the unique identifier of a font to the font management system in ATSUI.

```
typedef FMFont ATSUFontID;
```

Discussion
You pass the `ATSUFontID` type with functions that set and obtain font information. The `ATSUFontID` type is passed back by functions that count fonts installed on a user's system. The `ATSUFontID` type can be also used to set and get the font in a style run; see "Attribute Tags" (page 2030).

An `ATSUFontID` specifies a font family and instance. This value is not guaranteed to remain constant if the system is restarted. You should obtain the font's unique name and store that information in documents for which you need persistent font information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUGlyphInfo

Contains information about a glyph.

```
struct ATSUGlyphInfo {
    GlyphID glyphID;
    UInt16 reserved;
    UInt32 layoutFlags;
    UniCharArrayOffset charIndex;
    ATSUStyle style;
    Float32 deltaY;
    Float32 idealX;
    SInt16 screenX;
    SInt16 caretX;
};
```

Fields

`glyphID`
A glyph ID. This is unique to the associated font.

`reserved`
Reserved for Apple's use.

`layoutFlags`
The layout flags associated with this glyph.

`charIndex`
The index of the character in the Unicode character stream from which this glyph is derived.

`style`
An `ATSUStyle` value that specifies the style object associated with this glyph.

`deltaY`
The cross-stream shift value for this glyph.

`idealX`
The ideal with-stream offset from the origin of this layout.

`screenX`
The device-adjusted with-stream offset from the origin of this layout.

`caretX`
The position in device coordinates where a trailing caret for this glyph intersects the baseline.

Discussion

This data structure is used by ATSUI to return the glyph information associated with one glyph.

ATSUGlyphInfoArray

Contains text layout information for an array of glyphs.

```
struct ATSUGlyphInfoArray {
    ATSUTextLayout layout;
    ItemCount numGlyphs;
    ATSUGlyphInfo glyphs[1];
};
```

Fields

`layout`
An `ATSUTextLayout` value that specifies the text layout object associated with the glyphs.

`numGlyphs`
The number of glyphs associated with the text layout object.

`glyphs`
An array of glyph information structures.

Discussion
This data structure is used by ATSUI to return the glyph information associated with the glyphs in a text layout object.

ATSUGlyphSelector

Contains information that directs ATSUI to use a specific glyph instead of the one ATSUI normally derives.

```
struct ATSUGlyphSelector {
    GlyphCollection     collection;
    GlyphID             glyphID;
};
typedef struct ATSUGlyphSelector        ATSUGlyphSelector;
```

Fields
`collection`
A value that represents the collection of glyphs you want ATSUI to use. See "Glyph Collection Types" (page 2050) for possible values you can supply.

`glyphID`
A glyph ID value or a collection ID (CID) value. Supply a glyph ID when the collection type is `kGlyphCollectionGID`. Otherwise supply a CID.

Discussion
The `ATSUGlyphSelector` structure along with the attribute tag `kATSUGlyphSelectorTag` allow display of glyphs that do not have an explicit Unicode character. You can use the `kATSUGlyphSelectorTag` to access characters in fonts that otherwise would not be accessible. You can choose the variant glyph by font-specific glyph ID or CID. For more information on CID conventions, see go to http://www.adobe.com. You can get the variant glyph information from an input method through the Text Services Manager using the Carbon event key `kEventParamTextInputGlyphInfoArray`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeTypes.h`

ATSJustPriorityWidthDeltaOverrides

Contains justification width delta override structures, one for each priority-level override.

```
typedef ATSJustWidthDeltaEntryOverride ATSJustPriorityWidthDeltaOverrides[4];
```

Discussion
For more information see `ATSJustWidthDeltaEntryOverride` (page 2010).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSLayoutTypes.h`

ATSJustWidthDeltaEntryOverride

Contains values that specify the amount of space that can be added to or removed from the right and left sides of each of the glyphs of a given justification priority.

```
struct ATSJustWidthDeltaEntryOverride {
    Fixed beforeGrowLimit;
    Fixed beforeShrinkLimit;
    Fixed afterGrowLimit;
    Fixed afterShrinkLimit;
    JustificationFlags growFlags;
    JustificationFlags shrinkFlags;
};
typedef struct ATSJustWidthDeltaEntryOverride ATSJustWidthDeltaEntryOverride;
```

Fields

`beforeGrowLimit`
The proportion by which a glyph can expand on the left side (top side for vertical text). For example, a value of 0.2 means that a 24-point glyph can have by no more than 4.8 points (0.2 x 24 = 4.8) of extra space added on the left side (top side for vertical text).

`beforeShrinkLimit`
The proportion by which a glyph can shrink on the left side (top side for vertical text). If specified, this value should be negative.

`afterGrowLimit`
The proportion by which a glyph can expand on the right side (bottom side for vertical text).

`afterShrinkLimit`
The proportion by which a glyph can shrink on the right side (bottom side for vertical text). If specified, this value should be negative.

`growFlags`
Mask constants that indicate whether ATSUI should apply the limits defined in the `beforeGrowLimit` and `afterGrowLimit` fields. See "Justification Override Mask Constants" in the Font Manager for a description of possible values. These mask constants also control whether unlimited gap absorption should be applied to the priority of glyphs specified in the given width delta override structure. You can use these mask constants to selectively override the grow case only, while retaining default behavior for other cases.

`shrinkFlags`
Mask constants that indicate whether ATSUI should apply the limits defined in the `beforeShrinkLimit` and `afterShrinkLimit` fields. See "Justification Override Mask Constants" in the Font Manager for a description of possible values. These mask constants also control whether unlimited gap absorption should be applied to the priority of glyphs specified in the given width delta override structure. You can use these mask constants to selectively override the shrink case only, while retaining default behavior for other cases.

Discussion

The `JustWidthDeltaEntryOverride` structure specifies proportions for justification growth and shrinkage, both on the left and the right sides. The growth and shrinkage values override the font-specified widths, such as those specified by the font for kashidas.

It also contains justification flags. The `ATSJustWidthDeltaEntryOverride` data type can be used to set and get justification behavior and priority override weighting; see "Attribute Tags" (page 2030).

If you need to access other `'just'` table constants and structures from the `'sfnt'` resource, see the header file SFNTLayoutTypes.h.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSLayoutTypes.h`

ATSULayoutOperationOverrideSpecifier

Contains an layout operation selector and a pointer to a layout operation override callback.

```
struct ATSULayoutOperationOverrideSpecifier {
    ATSULayoutOperationSelector  operationSelector;
    ATSUDirectLayoutOperationOverrideUPP  overrideUPP;
};
typedef struct ATSULayoutOperationOverrideSpecifier
ATSULayoutOperationOverrideSpecifier;
```

Fields

`operationSelector`

A layout operation selector that specifies the operation for which the callback should be invoked. See "Layout Operation Selectors" (page 2055) for the selectors you can specify.

`overrideUPP`

A universal procedure pointer to a layout operation override callback.

Discussion
You can pass this structure as an attribute value for the layout attribute tag `kATSULayoutOperationOverrideTag`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSLayoutTypes.h`

ATSULineRef

Represents a reference to a structure that specifies a line of text.

```
typedef struct ATSGlyphVector *ATSULineRef;
```

Discussion
You get an ATSUI line reference from ATSUI when your layout operation override callback is invoked. The line reference refers to the line that ATSUI is in the process of laying out.

From within your callback, you pass an ATSUI line reference to the function `ATSUDirectGetLayoutDataArrayPtrFromLineRef` to obtain layout data for that line. The only way you can obtain an ATSUI line reference is from inside your layout operation override callback. An ATSUI line reference is not valid is outside of the callback.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSLayoutTypes.h`

ATSUStyle

Represents a reference to an opaque structure that contains information about a style object.

```
typedef struct OpaqueATSUStyle *ATSUStyle;
```

Discussion
A style object is an opaque structure encapsulating the following character-level style settings

- style attributes: including font ID, font size, font color, kerning control, optical alignment, verticality, and with-stream (left-right) and cross-stream (up-down) shifting (as for superscripts and subscripts)
- font features: including ligatures, swashes, and alternate glyph forms
- font variations: such as continually varying font weight, width, or slant

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSUStyleRunInfo

Contains information for a style run.

```
struct ATSUStyleRunInfo {
    UniCharCount        runLength;
    ItemCount           styleObjectIndex;
};
typedef struct ATSUStyleRunInfo         ATSUStyleRunInfo;
```

Fields
`runLength`
The length of the style run.
`styleObjectIndex`
An index into an array of unique style objects.

Discussion
This structure is used by the function `ATSUUnflattenStyleRunsFromStream` (page 1972) to return style run information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSUTab

Contains tab settings.

```
struct ATSUTab {
    ATSUTextMeasurement  tabPosition;
    ATSUTabType                         tabType;
};
typedef struct ATSUTab ATSUTab;
```

Fields

`tabPosition`

Specifies a tab position.

`tabType`

Specifies a type of tab stop. See "Tab Positioning Options" (page 2066).

Discussion

You can set tabs for a text layout object by calling the function `ATSUSetTabArray` (page 1962). You can obtain tab settings by calling the function `ATSUGetTabArray` (page 1918).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSUnicodeTypes.h`

ATSUTextLayout

Represents a reference to an opaque text layout structure that contains information about a text layout.

```
typedef struct OpaqueATSUTextLayout* ATSUTextLayout;
```

Discussion

The basic building block upon which ATSUI operates is a text layout object (`ATSUTextLayout`). A text layout object ties one or more paragraphs of text together with style attributes that may apply to characters, lines, or the entire layout. The text layout object itself contains information about line and layout attributes, including justification, rotation, direction, and others. Character style information is contained in a style object, which is only associated with, not contained by, a text layout object. For more information on text layout objects, see *Inside Mac OS X: Rendering Unicode Text With ATSUI*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ATSUnicodeTypes.h`

ATSUTextMeasurement

Represents measurements needed by ATSUI to lay out text, such as outline metrics and line width, ascent, descent.

```
typedef Fixed ATSUTextMeasurement;
```

Discussion

The `ATSUTextMeasurement` type is defined as a `Fixed` value, with a limit of 32K. You must ensure that your measurements are converted to `Fixed` values before passing them to ATSUI functions that use this type.

ATSUI uses fractional `Fixed` values instead of `short` values used in QuickDraw Text. Fractional `Fixed` values provide exact outline metrics and line specifications such as line width, ascent, descent, and so on.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

ATSTrapezoid

Contains the coordinates of the typographic bounding trapezoid for the final layout of a line a text.

```
struct ATSTrapezoid {
    FixedPoint upperLeft;
    FixedPoint upperRight;
    FixedPoint lowerRight;
    FixedPoint lowerLeft;
};
```

Fields

`upperLeft`
A structure of type `FixedPoint` that contains the upper left coordinates (assuming a horizontal line of text) of the typographic glyph bounds.

`upperRight`
A structure of type `FixedPoint` that contains the upper right coordinates (assuming a horizontal line of text) of the typographic glyph bounds.

`lowerRight`
A structure of type `FixedPoint` that identifies the lower right coordinates (assuming a horizontal line of text) of the typographic glyph bounds.

`lowerLeft`
A structure of type `FixedPoint` that identifies the lower left coordinates (assuming a horizontal line of text) of the typographic glyph bounds.

Discussion
The dimensions of the resulting trapezoid are relative to the coordinates specified in the `iTextBasePointX` and `iTextBasePointY` parameters. The width of the glyph bounds is determined based on the value passed in the `iTypeOfBounds` parameter.

The function `ATSUGetGlyphBounds` (page 1904) passes back an array of structures of type `ATSTrapezoid` to specify the enclosing trapezoid(s) of a final laid-out line of text. If the range of text spans directional boundaries, `ATSUGetGlyphBounds` produces multiple trapezoids defining these regions.

Version Notes
In ATSUI 1.1, the function `ATSUGetGlyphBounds` can pass back a maximum of 31 bounding trapezoids. In ATSUI 1.2, `ATSUGetGlyphBounds` can pass back as many as 127 bounding trapezoids.

ATSUUnhighlightData

Contains data needed to redraw the background.

```
struct ATSUUnhighlightData {
    ATSUBackgroundDataType dataType;
    ATSUBackgroundData unhighlightData;
};
```

Fields

`dataType`

The data type of the background—a color or a callback.

`unhighlightData`

A background color or a universal procedure pointer to a callback that redraws the background.

USTL Data Structure Data Types

The data types in this section define the `'ustl'` data structure, which is the data structure used by ATSUI to contain flattened data. The `'ustl'` data structure has four blocks. The Block 1 structure defines is a header for the entire `'ustl'` data structure. Block 2 structures define flattened text layout data. (Note that Block 2 structures are not currently used by the functions `ATSUFlattenStyleRunsToStream` (page 1882) and `ATSUUnflattenStyleRunsFromStream` (page 1972).) Block 3 structures define flattened style run data. Block 4 structures define flattened style data.

The `'ustl'` data structure can accommodate any ATSUI text layout and style run data associated with a document. That is, the `'ustl'` data structure can contain data for multiple text layout objects, multiple style runs, and multiple style objects. Within each block (text layout, style run, and style) you must specify the number structures in that block.

ATSFlatDataMainHeaderBlock

Contains the `'ustl'` data structure version and size and provides offsets to the text layout, style run, and style list data blocks.

```
struct ATSFlatDataMainHeaderBlock {
    UInt32              version;
    ByteCount           sizeOfDataBlock;
    ByteCount           offsetToTextLayouts;
    ByteCount           offsetToStyleRuns;
    ByteCount           offsetToStyleList;
};
typedef struct ATSFlatDataMainHeaderBlock ATSFlatDataMainHeaderBlock;
```

Fields

`version`

The version number of the `'ustl'` data structure. You must make sure this number is the first item in the data block, otherwise the data may not be readable by code written to parse earlier versions of `'ustl'` data.

`sizeOfDataBlock`

The total size of the data in bytes, including the four bytes needed for the version number.

`offsetToTextlayouts`

The offset from the beginning of the data block to the flattened text layout data. You can set this value to `0` if there is no text layout data. This value specifies the offset to the `ATSFlatDataTextLayoutDataHeader` (page 2016) structure.

`offsetToStyleRuns`

The offset from the beginning of the data to the flattened style run data. You can set this value to `0` if there is no flattened style run data. This value specifies the offset to the `ATSFlatDataStyleRunDataHeader` (page 2020) structure.

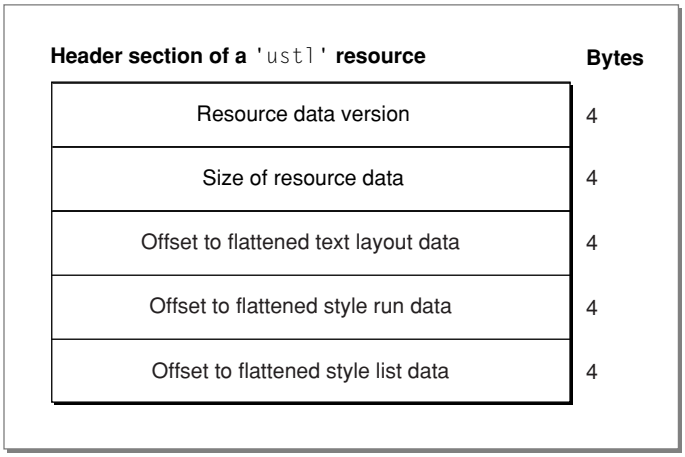
`offsetToStyleList`

The offset to the flattened style list data. You can set this value to `0` if there is no flattened style list data. This value specifies the offset to the `ATSFlatDataStyleListHeader` (page 2021) structure.

Discussion

The structure `ATSFlatDataMainHeaderBlock` is Block 1 of the `'ustl'` data structure. This structure contains information about the rest of the `'ustl'` data structure and provides offsets to each of the other three data blocks. Figure 46-1 illustrates the main header structure.

Figure 46-1 The main header for the ustl data structure



Per the `'ustl'` specification, all data blocks with the `'ustl'` data structure must maintain 4-byte alignment. For such items as font names, which have a variable width, you must add padding bytes to ensure the 4-byte alignment is always maintained.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSUnicodeFlattening.h`

ATSFlatDataTextLayoutDataHeader

Contains size, length, and offset information for a text layout data block.

```
struct ATSFlatDataTextLayoutDataHeader {
        ByteCount           sizeOfLayoutData;
        ByteCount           textLayoutLength;
        ByteCount           offsetToLayoutControls;
        ByteCount           offsetToLineInfo;
};
typedef struct ATSFlatDataTextLayoutDataHeader ATSFlatDataTextLayoutDataHeader;
```

Fields

`sizeOfLayoutData`

The size of the flattened text layout data. This value must include any bytes that have been added to maintain the required 4-byte alignment.

`textLayoutLength`

The number of characters to which the flattened text layout data applies.

`offsetLayoutControls`

The offset to the flattened layout control data. This offset is relative to the start of the text layout data block, and specifies the offset to the `ATSFlatDataLayoutControlsDataHeader` (page 2019) structure. The offset can be set to zero if there are no layout controls.

`offsetToLineLength`

The offset to the flattened line info data. This offset is relative to the start of the text layout data block, and specifies the offset to the `ATSFlatDataLineInfoHeader` (page 2019) structure. The offset can be set to zero if there is no line info in this layout.

Discussion

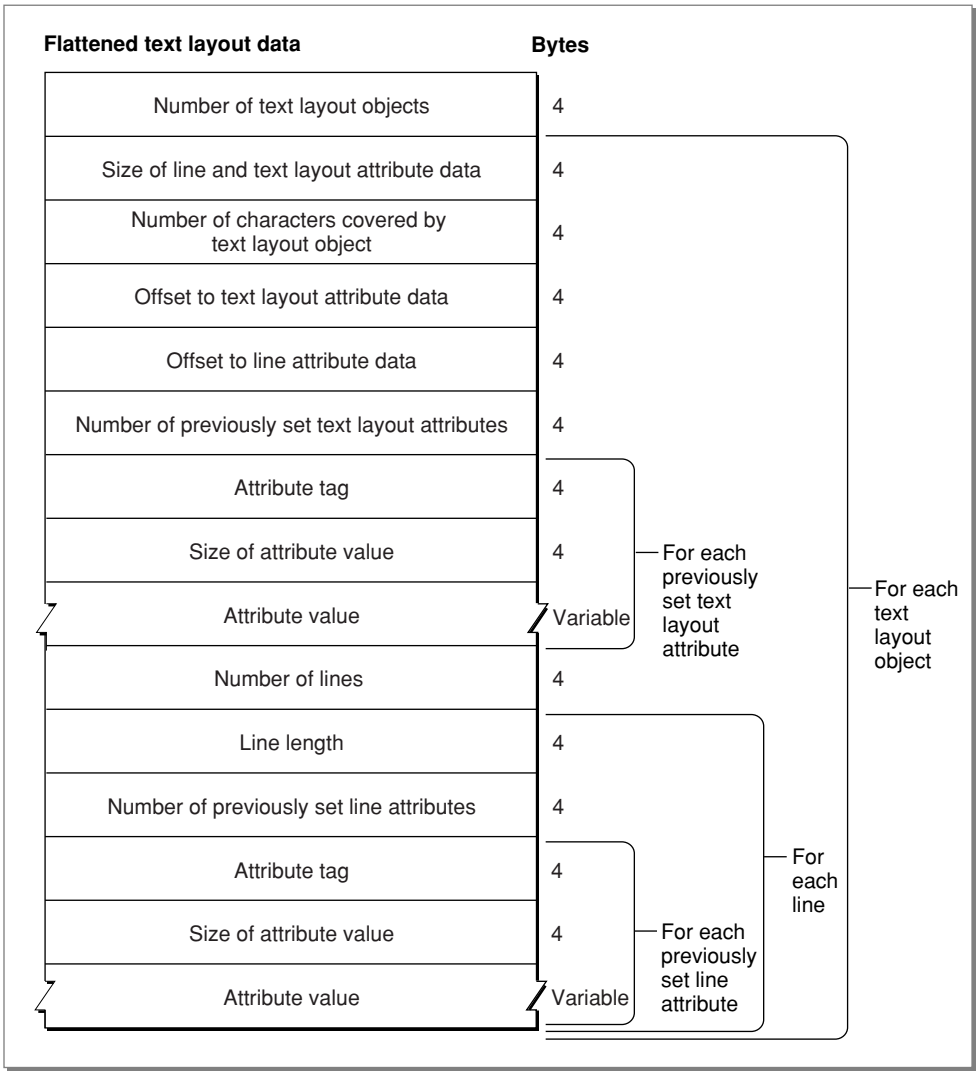
The `ATSFlatDataTextLayoutDataHeader` structure is a block 2 data structure and it is the main header for text layout data. If you have text layout data to flatten or unflatten, you must have one of these structures. for each text layout object whose data you want to flatten.

Note that the `ATSFlatDataTextLayoutDataHeader` data structure(s) must be preceded by an `ItemCount` value that specifies the number of `ATSFlatDataTextLayoutDataHeader` data structures included in the flattened data. Although the `ItemCount` value is not part of any `'ustl'` data structure, you need to include this 4-byte value when you flatten your text layout data so that you can successfully parse the flattened data at a later time.

The `offsetToTextLayouts` field in the `ATSFlatDataMainHeaderBlock` (page 2015) structure specifies the offset to the structure `ATSFlatDataTextLayoutDataHeader`.

Figure 46-2 depicts the flattened text layout data. At the top of the figure is the information contained in the data header (`ATSFlatDataTextLayoutDataHeader`). Following the header are layout controls data (see `ATSFlatDataLayoutControlsDataHeader` (page 2019)) and line length data (see `ATSFlatDataLineInfoHeader` (page 2019) and `ATSFlatDataLineInfoData` (page 2020)).

Figure 46-2 Flattened text layout data



If the `offsetToLayoutControls` value is not zero, there must be a `ATSFlatDataLayoutControlsDataHeader` (page 2019) structure that contains a count of the number of layout controls and an array of layout control attribute data.

If the `offsetToLineInfo` is not zero, then following the flattened layout controls data you must have an `ATSFlatDataLineInfoHeader` (page 2019) structure.

This and other Block 2 structures are not currently used by the functions `ATSUFlattenStyleRunsToStream` (page 1882) and `ATSUUnflattenStyleRunsFromStream` (page 1972).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSFlatDataLayoutControlsDataHeader

Contains the number of flattened layout controls and an array of layout control attribute data.

```
struct ATSFlatDataLayoutControlsDataHeader {
        ItemCount           numberOfLayoutControls;
        ATSUAttributeInfo   controlArray[1];
};typedef struct ATSFlatDataLayoutControlsDataHeader
ATSFlatDataLayoutControlsDataHeader;
```

Fields

`numberOfLayoutControls`

The number of flattened layout controls. There should be at least one layout control that specifies the line direction of the layout.

`controlArray[1]`

The first entry in an array of ATSUI attribute information. There should be `numberOfLayoutControls` entries in this array. If necessary, each ATSUI attribute info structure in the array should be followed by padding bytes to maintain the required 4-byte alignment. The value in the `fValueSize` field of each `ATSUAttributeInfo` structure must specify the size of the attribute value, and must not reflect any padding bytes you added.

Discussion

The `ATSFlatDataLayoutControlsDataHeader` structure is the header for the flattened layout controls structure. The `offsetToLayoutControls` field in the `ATSFlatDataTextLayoutDataHeader` (page 2016) structure specifies the offset to the structure `ATSFlatDataLayoutControlsDataHeader`. If there are no layout controls, you do not need the `ATSFlatDataLayoutControlsDataHeader` structure.

This and other Block 2 structures are not currently used by the functions `ATSUFlattenStyleRunsToStream` (page 1882) and `ATSUUnflattenStyleRunsFromStream` (page 1972).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSUnicodeFlattening.h`

ATSFlatDataLineInfoHeader

Contains the number of lines and an array of line information data.

```
struct ATSFlatDataLineInfoHeader {
        ItemCount           numberOfLines;
        ATSFlatDataLineInfoData  lineInfoArray[1];
};
typedef struct ATSFlatDataLineInfoHeader ATSFlatDataLineInfoHeader;
```

Fields

`numberOfLines`

The number of flattened line info structures that are stored in this block. This value should be greater than zero and equal to the number of soft line breaks in the layout plus one.

`lineInfoArray[1]`

The first entry in a array of `ATSFlatDataLineInfoData` (page 2020) structures. There should be `numberOfLines` entries in this array.

Discussion
The `ATSFlatDataLineInfoHeader` structure is the main data header for the flattened line info data. The value `offsetToLineInfo` in the `ATSFlatDataTextLayoutDataHeader` (page 2016) specifies the offset to the `ATSFlatDataLineInfoHeader` structure.

This and other Block 2 structures are not currently used by the functions `ATSUFlattenStyleRunsToStream` (page 1882) and `ATSUUnflattenStyleRunsFromStream` (page 1972).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSFlatDataLineInfoData

Contains a line length and the number of line controls for a line of flattened text.

```
struct ATSFlatDataLineInfoData {
        UniCharCount        lineLength;
        ItemCount           numberOfLineControls;
};
typedef struct ATSFlatDataLineInfoData  ATSFlatDataLineInfoData;
```

Fields

`lineLength`
The number of `UniChars` characters in the line.

`numberOfLineControls`
The number of line controls applied to the line. You can set this value to zero if there are no line controls applied to this line.

Discussion
If the `numberOfLineControls` is not zero, then you must supply an array of `ATSUAttributeInfo` structures that contains `numberOfLineControls` elements.

This and other Block 2 structures are not currently used by the functions `ATSUFlattenStyleRunsToStream` (page 1882) and `ATSUUnflattenStyleRunsFromStream` (page 1972).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSFlatDataStyleRunDataHeader

Contains the number of style runs and style run information for the style run data block.

```
struct ATSFlatDataStyleRunDataHeader {
    ItemCount           numberOfStyleRuns;
    ATSUStyleRunInfo    styleRunArray[1];
};
typedef struct ATSFlatDataStyleRunDataHeader ATSFlatDataStyleRunDataHeader;
```

Fields

`numberOfStyleRuns`

The number of style run data structures stored in this block.

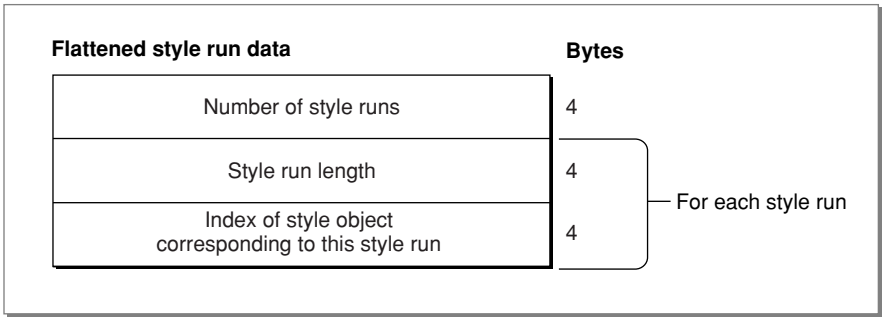
`styleRunArray[1]`

The first entry in a array of `ATSUStyleRunInfo` structures. There should be `numberOfStyleRuns` entries in this array.

Discussion

The `ATSFlatDataStyleRunDataHeader` structure precedes style run data structures. The `offsetToStyleRuns` field in the `ATSFlatDataMainHeaderBlock` (page 2015) specifies the offset to the structure `ATSFlatDataStyleRunDataHeader`.

Figure 46-3 Flattened style run data



This is a Block 3 structure. Block 3 structures are used by ATSUI style run flattening and parsing functions, `ATSUFlattenStyleRunsToStream` (page 1882) and `ATSUUnflattenStyleRunsFromStream` (page 1972), to represent flattened style run information. These structures work together with Block 4 structures.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSUnicodeFlattening.h`

ATSFlatDataStyleListHeader

Contains the number of styles and the first item in the style list style data header.

```
struct ATSFlatDataStyleListHeader {
        ItemCount           numberOfStyles;
        ATSFlatDataStyleListStyleDataHeader  styleDataArray[1];
};
typedef struct ATSFlatDataStyleListHeader ATSFlatDataStyleListHeader;
```

Fields

`numberOfStyles`

The number of flattened style objects in this block.

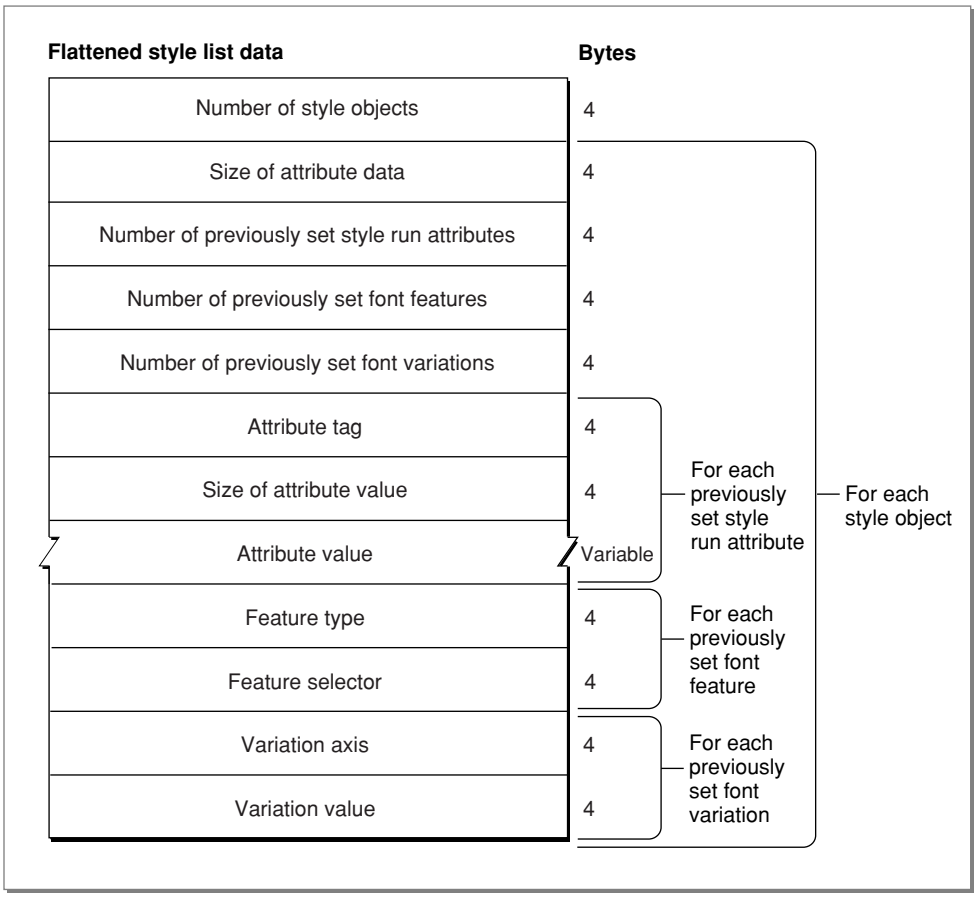
`styleDataArray[1]`

The first item in an array of `ATSFlatDataStyleListStyleDataHeader` (page 2023) structures. There should be `numberOfStyles` entries in this array. Note that the data stored in these structures can be of variable sizes.

Discussion

The `ATSFlatDataStyleListHeader` structure is the main header for Block 4. The `offsetToStyleList` field in the `ATSFlatDataMainHeaderBlock` (page 2015) specifies the offset to the structure `ATSFlatDataStyleListHeader`.

Figure 46-4 Flattened style list data



Availability

Available in Mac OS X v10.2 and later.

Declared In
ATSUnicodeFlattening.h

ATSFlatDataStyleListStyleDataHeader

Contains size information and the number of attributes, features, and variations for the style list data block.

```
struct ATSFlatDataStyleListStyleDataHeader {
            ByteCount           sizeOfStyleInfo;
            ItemCount           numberOfSetAttributes;
            ItemCount           numberOfSetFeatures;
            ItemCount           numberOfSetVariations;
};
typedef struct ATSFlatDataStyleListStyleDataHeader
ATSFlatDataStyleListStyleDataHeader;
```

Fields

`sizeOfStyleInfo`
The size of the flattened style object. This value should include the four bytes for this field (`sizeOfStyleInfo`) and any padding bytes you add to end of the structure to maintain the required 4-byte alignment.

`numberOfSetAttributes`
The number of attributes in the flattened style object. You should have at least one attribute for the font data, although you can set this value to `0` if you do not want to specify font data.

`numberOfSetFeatures`
The number of font features in the flattened style object. You can set this value to `0` if there are no font features in the style object.

`numberOfSetVariations`
The number of font variations in the flattened style object. You can set this value to `0` if there are no font variations in the style object.

Discussion
The `ATSFlatDataStyleListStyleDataHeader` structure forms the beginning of an individually flattened `ATSUStyle` object. This structure precedes the following data:

1. If the value `numberOfSetAttributes` is non-zero, there must be an array of `ATSUAttributeInfo` structures immediately following the `ATSFlatDataStyleListStyleDataHeader` structure to store the style attributes. This is a variable-size array. The number of `ATSUAttributeInfo` structures must be equal to the value `numberOfSetAttributes`, one structure for each attribute.

   If the value `numberOfSetAttributes` is zero, you do not need an array of `ATSUAttributeInfo` structures.

2. If the value `numberOfSetFeatures` is non-zero, there must be an array of `ATSFlatDataStyleListFeatureData` structures. These structures must appear immediately following the `ATSUAttributeInfo` array above (if there is such an array). The number of `ATSFlatDataStyleListFeatureData` structures must be equal to the value `numberOfSetFeatures`, one structure for each feature.

   If the value `numberOfSetFeatures` is zero, you do not need an array of `ATSFlatDataStyleListFeatureData` structures.

3. If the value `numberOfSetVariations` is non-zero, there must be an array of `ATSFlatDataStyleListVariationData` structures immediately following the `ATSFlatDataStyleListFeatureData` array (if there is such an array). The number of `ATSFlatDataStyleListVariationData` structures must be equal to the value `numberOfSetVariations`, one structure for each variation.

This is a Block 4 structure. Block 4 structures store flattened `ATSUStyle` objects and are currently used by the ATSUI style run flattening and parsing functions, `ATSUFlattenStyleRunsToStream` (page 1882) and `ATSUUnflattenStyleRunsFromStream` (page 1972).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSFlatDataStyleListFeatureData

Contains flattened font feature data.

```
struct ATSFlatDataStyleListFeatureData {
            ATSUFontFeatureType  theFeatureType;
            ATSUFontFeatureSelector  theFeatureSelector;
};
typedef struct ATSFlatDataStyleListFeatureData ATSFlatDataStyleListFeatureData;
```

Fields

`theFeatureType`
A font feature type.

`theFeatureSelector`
A font feature selector.

Discussion
This is a Block 4 structure. The structure `ATSFlatDataStyleListFeatureData` stores flattened font feature data. If the value `numberOfSetFeatures` in the `ATSFlatDataStyleListStyleDataHeader` (page 2023) structure is non-zero, an array of these structure must follow the array of font data attributes (if such an array exists) if the `numberOfSetFeatures` is non-zero. The number of `ATSFlatDataStyleListFeatureData` structures must be equal to the value `numberOfSetFeatures`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSFlatDataStyleListVariationData

Contains flattened font variation axis data.

```
struct ATSFlatDataStyleListVariationData {
            ATSUFontVariationAxis  theVariationAxis;
            ATSUFontVariationValue  theVariationValue;
};
typedef struct ATSFlatDataStyleListVariationData ATSFlatDataStyleListVariationData;
```

Fields

`theVariationAxis`

A font variation axis.

`theVariationValue`

A font variation value.

Discussion

This is a Block 4 structure. The structure `ATSFlatDataStyleListVariationData` stores flattened font variation data. If the value `numberOfSetVariations` in the `ATSFlatDataStyleListStyleDataHeader` (page 2023) structure is non-zero, an array of these structure must follow the array of font features (if such an array exists) if the `numberOfSetVariations` is non-zero. The number of `ATSFlatDataStyleListVariationData` structures must be equal to the value `numberOfSetVariations`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSUnicodeFlattening.h`

ATSFlatDataFontNameDataHeader

Contains font name information.

```
struct ATSFlatDataFontNameDataHeader {
        ATSFlatDataFontSpeciferType  nameSpecifierType;
        ByteCount           nameSpecifierSize;
};
typedef struct ATSFlatDataFontNameDataHeader ATSFlatDataFontNameDataHeader;
```

Fields

`nameSpecifierType`

A font specifier for the type of the font name data you plan to supply. See "Flattened Data Font Type Selectors" (page 2046) for a list of the font specifiers you can supply. The font name data must follow the `ATSFlatDataFontNameDataHeader` structure.

`nameSpecifierSize`

The size of the flattened font name data. This value must not include any padding bytes that may be necessary to achieve the required 4-byte alignment, unless the padding bytes are specified as part of structure, such as with the `ATSFlatDataFontSpecRawNameData` structure.

Discussion

Font information can be recorded in an `ATSUStyle` object using the attribute tag `kATSUFontTag` and an attribute value that is of type `ATSUFontID`. Unfortunately, a font ID can vary between systems or system startups, which means you cannot ensure that the font used when the style is flattened is the same font that will be used with the style is unflattened. To preserve font information, you must flatten font name data. You specify font information using the structure `ATSFlatDataFontNameDataHeader`. You store this structure as a style attribute value. You must make sure this structure maintains the required 4-byte alignment.

Following the `ATSFlatDataFontNameDataHeader` structure must be the flattened font name data of the type specified by the `nameSpecifierType` field. For instance, if the value of the `nameSpecType` field is `kATSFlattenedFontNameSpecifierRawNameData`, the structure that immediately follows should be a `ATSFlatDataFontSpecRawNameDataHeader` (page 2026) structure.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSFlatDataFontSpecRawNameDataHeader

Contains raw font name data.

```
struct ATSFlatDataFontSpecRawNameDataHeader {
            ItemCount           numberOfFlattenedNames;
            ATSFlatDataFontSpecRawNameData  nameDataArray[1];
};
typedef struct ATSFlatDataFontSpecRawNameDataHeader
ATSFlatDataFontSpecRawNameDataHeader;
```

Fields

`numberOfFlattenedNames`
The number of flattened font names. There must be at least one flattened font name, otherwise the structure is malformed.

`nameDataArray[1]`
The first element in an array of raw font name data.

Discussion
The function `ATSUUnflattenStyleRunsFromStream` (page 1972) searches for fonts that match the font data provides in the `nameDataArray`. ATSUI obtains matches for all the font name specifiers in the structure. You must supply at least one entry in the `nameDataArray`, but you may want to supply more than one entry to ensure a specific match. For example, the default ATSUI implementation is to use two name specifiers—the full name of the font (`kFontFullName`) and the font manufacturer's name (`kFontManufacturerName`).

The `ATSFlatDataFontSpecRawNameDataHeader` structure must be followed by one or more `ATSFlatDataFontSpecRawNameData` structures. The number of structures must match the value specified by the `numberOfFlattenedName` field.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSUnicodeFlattening.h`

ATSFlatDataFontSpecRawNameData

Contains data for a font name.

```
struct ATSFlatDataFontSpecRawNameData {
            FontNameCode        fontNameType;
            FontPlatformCode    fontNamePlatform;
            FontScriptCode      fontNameScript;
            FontLanguageCode    fontNameLanguage;
            ByteCount           fontNameLength;
};
typedef struct ATSFlatDataFontSpecRawNameData ATSFlatDataFontSpecRawNameData;
```

Fields

`fontNameType`

The type of font name. You must supply this parameter.

`fontNamePlatform`

The platform type of the font name. You should specify this if you know it (Unicode, Mac, and so forth). If you do not know the platform type, then specify `kFontNoPlatform`. In this case all matching is done by ATSUI based on the first font in the name table that matches the other parameters in this structure.

`fontNameScript`

The script code of the font name based on the platform specified in the `fontNamePlatform` field. If you set this to `kFontNoScript`, the name is matched based on the first font in the name table that matches the other font name parameters in this structure.

`fontNameLanguage`

The language of the font name. If you set this to `kFontNoLanguage`, the name is matched based on the first font in the name table that matches the other font name parameters in this structure.

`fontNameLength`

The length of the font name. The length should include any padding bytes needed to maintain the required 4-byte alignment.

Discussion

The `ATSFlatDataFontSpecRawNameData` structure is the structure in which raw font name data is actually stored. This structure is used only when the value of the `nameSpecifierType` field in the `ATSFlatDataFontNameDataHeader` (page 2025) structure is `kATSFlattenedFontSpecifierRawNameData`. The structure stores multiple font name table entries for the purposes of reconstructing an `ATSUFontID` value for the same font at some time in the future.

When the ATSUI parsing function `ATSUUnflattenStyleRunsFromStream` searches for fonts to match the font data in this structure, it obtains matches for all the font name specifiers in the structure. The default ATSUI implementation is to use two name specifiers—the full name of the font (`kFontFullName`) and the font manufacturer's name (`kFontManufacturerName`).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ATSUnicodeFlattening.h`

Universal Procedure Pointers

ATSUDirectLayoutOperationOverrideUPP

Defines a universal procedure pointer to a layout operation callback.

```
typedef ATSUDirectLayoutOperationOverrideProcPtr
ATSUDirectLayoutOperationOverrideUPP;
```

Discussion
For more information, see the description of the `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ATSLayoutTypes.h`

ATSCubicClosePathUPP

Defines a universal procedure pointer to a cubic close-path callback.

```
typedef ATSCubicClosePathProcPtr ATSCubicClosePathUPP;
```

Discussion
For more information, see the description of the `ATSCubicClosePathProcPtr` (page 1990) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSCubicCurveToUPP

Defines a universal procedure pointer to a cubic curve-to callback.

```
typedef ATSCubicCurveToProcPtr ATSCubicCurveToUPP;
```

Discussion
For more information, see the description of the `ATSCubicCurveToProcPtr` (page 1991) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSCubicLineToUPP

Defines a universal procedure pointer to a cubic line-to callback.

```
typedef ATSCubicLineToProcPtr ATSCubicLineToProcUPP;
```

Discussion
For more information, see the description of the `ATSCubicLineToProcPtr` (page 1992) callback function.

ATSCubicMoveToUPP

Defines a universal procedure pointer to a cubic move-to callback.

```
typedef ATSCubicMoveToProcPtr ATSCubicMoveToUPP;
```

Discussion
For more information, see the description of the `ATSCubicMoveToProcPtr` (page 1993) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticClosePathUPP

Defines a universal procedure pointer to a quadratic close-path callback.

```
typedef ATSQuadraticClosePathProcPtr ATSQuadraticClosePathUPP;
```

Discussion
For more information, see the description of the `ATSQuadraticClosePathProcPtr` (page 1994) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticCurveUPP

Defines a universal procedure pointer to a quadratic curve callback.

```
typedef ATSQuadraticCurveProcPtr ATSQuadraticCurveUPP;
```

Discussion
For more information, see the description of the `ATSQuadraticCurveProcPtr` (page 1995) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticLineUPP

Defines a universal procedure pointer to a quadratic line callback.

```
typedef ATSQuadraticLineProcPtr ATSQuadraticLineUPP;
```

Discussion
For more information, see the description of the `ATSQuadraticLineProcPtr` (page 1996) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

ATSQuadraticNewPathUPP

Defines a universal procedure pointer to a quadratic new-path callback.

```
typedef ATSQuadraticNewPathProcPtr ATSQuadraticNewPathUPP;
```

Discussion
For more information, see the description of the `ATSQuadraticNewPathProcPtr` (page 1997) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeGlyphs.h`

RedrawBackgroundUPP

Defines a universal procedure pointer to a redraw-background callback.

```
typedef RedrawBackgroundProcPtr RedrawBackgroundUPP;
```

Discussion
For more information, see the description of the `RedrawBackgroundProcPtr` (page 1999) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ATSUnicodeTypes.h`

Constants

Attribute Tags

Specify attributes that can be applied to a style object, a text layout object, or a line in a text layout object.

```
typedef UInt32 ATSUAttributeTag;
enum {
    kATSULineWidthTag = 1L,
    kATSULineRotationTag = 2L,
    kATSULineDirectionTag = 3L,
    kATSULineJustificationFactorTag = 4L,
    kATSULineFlushFactorTag = 5L,
    kATSULineBaselineValuesTag = 6L,
    kATSULineLayoutOptionsTag = 7L,
    kATSULineAscentTag = 8L,
    kATSULineDescentTag = 9L,
    kATSULineLangRegionTag = 10L,
    kATSULineTextLocatorTag = 11L,
    kATSULineTruncationTag = 12L,
    kATSULineFontFallbacksTag = 13L,
    kATSULineDecimalTabCharacterTag = 14L,
    kATSULayoutOperationOverrideTag = 15L,
    kATSULineHighlightCGColorTag  = 17L,
    kATSUMaxLineTag = 18L,
    kATSULineLanguageTag = 10L,
    kATSUCGContextTag = 32767L,
    kATSUQDBoldfaceTag = 256L,
    kATSUQDItalicTag = 257L,
    kATSUQDUnderlineTag = 258L,
    kATSUQDCondensedTag = 259L,
    kATSUQDExtendedTag = 260L,
    kATSUFontTag = 261L,
    kATSUSizeTag = 262L,
    kATSUColorTag = 263L,
    kATSULangRegionTag = 264L,
    kATSUVerticalCharacterTag = 265L,
    kATSUImposeWidthTag = 266L,
    kATSUBeforeWithStreamShiftTag = 267L,
    kATSUAfterWithStreamShiftTag = 268L,
    kATSUCrossStreamShiftTag = 269L,
    kATSUTrackingTag = 270L,
    kATSUHangingInhibitFactorTag = 271L,
    kATSUKerningInhibitFactorTag = 272L,
    kATSUDecompositionFactorTag = 273L,
    kATSUBaselineClassTag = 274L,
    kATSUPriorityJustOverrideTag = 275L,
    kATSUNoLigatureSplitTag = 276L,
    kATSUNoCaretAngleTag = 277L,
    kATSUSuppressCrossKerningTag = 278L,
    kATSUNoOpticalAlignmentTag = 279L,
    kATSUForceHangingTag = 280L,
    kATSUNoSpecialJustificationTag = 281L,
    kATSUStyleTextLocatorTag = 282L,
    kATSUStyleRenderingOptionsTag = 283L,
    kATSUAscentTag         = 284L,
    kATSUDescentTag = 285L,
    kATSULeadingTag = 286L,
    kATSUGlyphSelectorTag = 287L,
    kATSURGBAlphaColorTag     = 288L,
    kATSUFontMatrixTag = 289L,
    kATSUStyleUnderlineCountOptionTag = 290L,
    kATSUStyleUnderlineColorOptionTag = 291L,
    kATSUStyleStrikeThroughTag    = 292L,
```

```
    kATSUStyleStrikeThroughCountOptionTag = 293L,
    kATSUStyleStrikeThroughColorOptionTag = 294L,
    kATSUStyleDropShadowTag = 295L,
    kATSUStyleDropShadowBlurOptionTag = 296L,
    kATSUStyleDropShadowOffsetOptionTag = 297L,
    kATSUStyleDropShadowColorOptionTag = 298L,
    kATSUMaxStyleTag = 299L,
    kATSULanguageTag = 264L,
    kATSUMaxATSUITagValue = 65535L
};
```

Constants

`kATSULineWidthTag`

Specifies the desired width of a line of text, in typographic points, of the line when drawn as justified or right-aligned text. The associated value is of type `ATSUTextMeasurement` (page 2013) and has a default value of `0`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineRotationTag`

Specifies the angle by which the entire line should be rotated. The associated value is a `Fixed` value that specifies degrees in a right-hand coordinate system, and has a default value of `0`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineDirectionTag`

Specifies a left-to-right or right-to-left direction for the glyphs in a text layout object, regardless of their natural direction as specified in the font. The associated value is `Boolean` (`kATSURightToLeftBaseDirection` or `kATSULeftToRightBaseDirection`) and has a default value of `GetSysDirection()`. See "Glyph Direction Selectors" (page 2051) for more information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineJustificationFactorTag`

Specifies how ATSUI should typographically fit a line of text to a given width (or height, in the case of vertical text). The associated value is a `Fract` value between 0 and 1 and has a default value of `kATSUNoJustification`. See "Line Justification Selectors" (page 2058) for information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineFlushFactorTag`

Specifies how ATSUI should place text in relation to one or both margins, which are the left and right sides (or top and bottom sides) of the text area. The associated value is a `Fract` value between 0 and 1 and has a default value of `kATSUStartAlignment`. See "Line Alignment Selectors " (page 2057) for information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineBaselineValuesTag`
Specifies the positions of different baseline types with respect to one another in a line of text. The associated value is of type `BslnBaselineRecord` and contains default values all of which are `0`. The values are calculated from other style attributes such as font and point size.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineLayoutOptionsTag`
Specifies how ATSUI should manipulate basic attributes of a line or the text layout object, such as whether a line should have optical hangers or whether the last line of a text layout object should be justified. The associated value is of type `ATSLineLayoutOptions` and has a default value of `kATSLineNoLayoutOptions`. See "Line Layout Attribute Tags" (page 2058) for information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineAscentTag`
Specifies line ascent. The associated value is of type `ATSUTextMeasurement` (page 2013) and has a default value of `kATSUseLineHeight`. See "Line Height and Font Tracking Selectors" (page 2057) for information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineDescentTag`
Specifies line descent. The associated value is of type `ATSUTextMeasurement` (page 2013) and has a default value of `kATSUseLineHeight`. See "Line Height and Font Tracking Selectors" (page 2057) for information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineLangRegionTag`
Specifies line language region. The associated value is a region code (see the Script Manager reference for a list of region codes) and has a default value of `kTextRegionDontCare`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineTextLocatorTag`
Specifies line text location. The associated value is of type `TextBreakLocatorRef` and has a default value of `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineTruncationTag`
Specifies where in a line truncation should occur. The associated value is of type `ATSULineTruncation` and has a default value of `kATSUTruncateNone`. See "Line Truncation Selectors" (page 2054) for the values that can be associated with the line truncation tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineFontFallbacksTag`
Specifies line font fallbacks. The associated value is of type `ATSUFontFallbacks` (page 2007). See "Font Fallback Methods" (page 2048) for information on the values that can be associated with this tag.

Available in Mac OS X v10.1 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineDecimalTabCharacterTag`
Specifies the current setting for the decimal separator, and affects the behavior of decimal tabs for a text layout (not an individual line). The associated value is of type `CFStringRef`. The CFString object (`CFStringRef`) is retained by the style object in which it is set. The default value is the user setting in System Preferences.

Declared in `ATSUnicodeTypes.h`.

Available in Mac OS X version 10.3 and later.

`kATSULineHighlightCGColorTag`
Specifies the current setting of the highlight color and opacity. The associated value is of type `CGColorRef`. This can be set as a line or layout control. The CGColor object (CGColorRef) is retained by the text layout object in which it is set.

Available in Mac OS X version 10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULayoutOperationOverrideTag`
Specifies to override a layout operation. The associated value is of type `ATSULayoutOperationOverrideSpecifier` and has a default value of `NULL`.

Available starting with Mac OS X version 10.2.

Declared in `ATSUnicodeTypes.h`.

`kATSUMaxLineTag`
A convenience tag that specifies the upper limit of the text layout attribute tags.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULineLanguageTag`
Not recommended. Instead use `kATSULineLangRegionTag`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUCGContextTag`
Specifies to use a Quartz context. When you use this tag to set up a Quartz context, ATSUI uses an 8-bit, sub-pixel rendering. This method of rendering positions glyph origins on fractional points, which results in superior rendering compared to ATSUI's default 4-bit pixel-aligned rendering. The attribute has a default value of `NULL`; you must provide a pointer to a `CGContext`. The `CGContext` is not retained by the text layout object; if the context is destroyed, the text layout contains an invalid `CGContext`. Available only in Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUQDBoldfaceTag`
Specifies a boldface text style. Text style attribute tags are included for compatibility with the `Style` type used by the QuickDraw function `TextFace`. If a font variant for this text style exists, ATSUI uses that variant. Otherwise, the variant is generated algorithmically. The associated value is of type `Boolean` and has a default value of `false`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUQDItalicTag`
Specifies an italic text style. Text style attribute tags are included for compatibility with the `Style` type used by the QuickDraw function `TextFace`. If a font variant for this text style exists, ATSUI uses that variant. Otherwise, the variant is generated. The associated value is of type `Boolean` and has a default value of `false`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUQDUnderlineTag`
Specifies an underline text style. Text style attribute tags are included for compatibility with the `Style` type used by the QuickDraw function `TextFace`. If a font variant for this text style exists, ATSUI uses that variant. Otherwise, the variant is generated. The associated value is of type `Boolean` and has a default value of `false`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUQDCondensedTag`
Specifies a condensed text style. Text style attribute tags are included for compatibility with the `Style` type used by the QuickDraw function `TextFace`. If a font variant for this text style exists, ATSUI uses that variant. Otherwise, the variant is generated. The associated value is of type `Boolean` and has a default value of `false`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUQDExtendedTag`
Specifies an extended text style. Text style attribute tags are included for compatibility with the `Style` type used by the QuickDraw function `TextFace`. If a font variant for this text style exists, ATSUI uses that variant. Otherwise, the variant is generated The associated value is of type `Boolean` and has a default value of `false`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUFontTag`
Specifies a unique value that identifies a font to the font management system. The associated value is of type `ATSUFontID` (page 2007) and has a default value of `GetScriptVariable (smSystemScript, smScriptAppFond)`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUSizeTag`

Specifies the font size of the text in the style run. The associated value, in typographic points (72 per inch), is of type `Fixed` and has a default value of `GetScriptVariable (smSystemScript, smScriptAppFondSize)`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUColorTag`

Specifies the color of the glyphs in a style run. The associated value is of type `RGBColor` and has a default value of `(0,0,0)`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULangRegionTag`

Specifies a language region. The associated value is a region code (see the Script Manager reference for a list of region codes) and has a default value of `GetScriptManagerVariable (smRegionCode)`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUVerticalCharacterTag`

Specifies which direction (vertical or horizontal) glyphs should be drawn. The associated value is of type `ATSUVerticalCharacterType` and has a default value of `kATSUStronglyHorizontal`. See "Vertical Character Types" (page 2068) for more information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUImposeWidthTag`

Specifies an imposed width. The associated value is of type `ATSUTextMeasurement` (page 2013) and has a default value of `0`; all glyphs use their own font defined advance widths.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUBeforeWithStreamShiftTag`

Specifies a uniform shift parallel to the baseline of the positions of individual pairs or sets of glyphs in the style run that's applied before (to the left) the glyphs of the style run. The associated value is of type `Fixed` and has a default value of `0`. Starting with Mac OS version 10.3, glyphs cannot be negatively shifted such that later glyphs appear before earlier glyphs. In other words, ATSUI limits the shift to a value that is, at most, the advance of the previous glyph.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUAfterWithStreamShiftTag`

Specifies a uniform shift parallel to the baseline of the positions of individual pairs or sets of glyphs in the style run that's applied after (to the right) the glyphs of the style run. The associated value is of type `Fixed` and has a default value of `0`. Starting with Mac OS version 10.3, glyphs cannot be negatively shifted such that later glyphs appear before earlier glyphs. In other words, ATSUI limits the shift to a value that is, at most, the advance of the current glyph.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUCrossStreamShiftTag`

Specifies the distance to raise or lower glyphs in the style run perpendicular to the text stream. This shift is vertical for horizontal text and horizontal for vertical text. The associated value (in points, 72 per inch) is of type `Fixed` and has a default value of `0`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUTrackingTag`

Specifies the relative proportion of font-defined adjustments to apply to interglyph positions. The associated value is of type `Fixed` and has a default value of `kATSNoTracking`. See "Line Height and Font Tracking Selectors" (page 2057) for information on the values that can be associated with this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUHangingInhibitFactorTag`

Specifies to what degree punctuation glyphs can hang beyond the end of a line for justification purposes. The associated value is a `Fract` value between `0` and `1` and has a default value of `0`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUKerningInhibitFactorTag`

Specifies how much to inhibit kerning; that is, the increase or decrease the space between glyphs. The associated value is a `Fract` value between `0` and `1` and has a default value of `0`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUDecompositionFactorTag`

Specifies the fractional adjustment to the font-specified threshold at which ligature decomposition occurs during justification. The associated value is a `Fract` value between `-1.0` and `1.0` and has a default value of `0` (no adjustment to the font-specified threshold).

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUBaselineClassTag`

Specifies the preferred baseline (such as Roman, hanging, or ideographic centered) to use for text of a given font in a style run. The associated value is of type `BslnBaselineClass` (see SFNTLayoutTypes.h) and has a default value of `kBSLNRomanBaseline`. You can set the value to `kBSLNNoBaselineOverride` to use intrinsic baselines.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUPriorityJustOverrideTag`

Specifies the degree to which ATSUI should override justification behavior for glyphs in the style run. The associated value is of type `ATSJustWidthDeltaEntryOverride` (page 2010). The default values in this structure are all `0`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUNoLigatureSplitTag`
Specifies whether or not ligatures and compound characters in a style have divisible components. The associated value is a `Boolean` and has a default value of `false`; ligatures and compound characters have divisible components.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUNoCaretAngleTag`
Specifies whether the text caret or edges of a highlighted area are always parallel to the slant of the style run's text or always perpendicular to the baseline. The associated value is a `Boolean` and has a default value of `false`; use the character's angularity to determine its boundaries.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUSuppressCrossKerningTag`
Specifies whether or not to suppress cross kerning. The associated value is a `Boolean` and has a default value of `false`; do not suppress automatic cross kerning (defined by font).

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUNoOpticalAlignmentTag`
Specifies the amount to which ATSUI should adjust glyph positions at the ends of lines to give a more even visual appearance to margins. The associated value is a `Boolean` and has a default value of `false`; do not suppress character's automatic optical positional alignment

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUForceHangingTag`
Specifies to treat glyphs in a style run as hanging punctuation, whether or not the font designer intended them to be. The associated value is a `Boolean` and has a default value of `false`; do not force the character's to hang beyond the line boundaries

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUNoSpecialJustificationTag`
Specifies whether processes (such as glyph stretching and ligature decomposition) that occur at the end of the justification process should be applied. The associated value is a `Boolean` and has a default value of `false`; perform post-compensation justification if needed

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStyleTextLocatorTag`
Specifies style text locator. The associated value is of type `TextBreakLocatorRef` and has a default value of `NULL`—region derived locator or the default Text Utilities locator.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStyleRenderingOptionsTag`
: Specifies style rendering options. The associated value is of type `ATSUStyleRenderingOptions` and has a default value of `kATSStyleApplyHints`—ATS glyph rendering uses hinting. See "Style Rendering Options" (page 2065) for more information on the values that can be associated with this tag.

  Available in Mac OS X v10.0 and later.

  Declared in `ATSUnicodeTypes.h`.

`kATSUAscentTag`
: Specifies the ascent value of a style's font. The associated value is of type `ATSUTextMeasurement` (page 2013) and has a default value of the ascent value of the style object's font with the current point size.

  Available starting with Mac OS X version 10.2.

  Declared in `ATSUnicodeTypes.h`.

`kATSUDescentTag`
: Specifies the descent value of a style's font. The associated value is of type `ATSUTextMeasurement` (page 2013) and has a default value of the descent value of the style object's font with the current point size. The leading value is not included as par of the descent.

  Declared in `ATSUnicodeTypes.h`.

  Available starting with Mac OS X version 10.2.

`kATSULeadingTag`
: Specifies the leading value of a style's font. The associated value is of type `ATSUTextMeasurement` (page 2013) and has a default value of the leading value of the style object's font with the current point size.

  Available starting with Mac OS X version 10.2.

  Declared in `ATSUnicodeTypes.h`.

`kATSUGlyphSelectorTag`
: Specifies a glyph collection. The associated value is an address to an `ATSUGlyphSelector` (page 2009) data structure. Using this tag allows you access to characters in the fonts that otherwise would not be accessible. You can choose the variant glyph by providing a font-specific glyph ID or a CID. For more information on CID conventions, see http://www.adobe.com. You can get the variant glyph information from an input method through the Text Services Manager using the Carbon event key, `kEventParamTextInputGlyphInfoArray`.

  Declared in `ATSUnicodeTypes.h`.

  Available starting with Mac OS X version 10.2.

`kATSURGBAlphaColorTag`
: Specifies RGB color with an alpha channel. The associated value is of type `ATSURGBAlphaColor` and has a default value of (0,0,0,1).

  Available starting with Mac OS X version 10.2.

  Declared in `ATSUnicodeTypes.h`.

`kATSUFontMatrixTag`

Specifies a font transformation matrix. The associated value is of type `CGAffineTransform`. (See the Quartz 2D reference documentation for more information on this data type.) You can use a font matrix to achieve effects through ATSUI at a style-run level that were previously available only by changing settings directly in a `CGContext`. When you use the tag `kATSUFontMatrixTag`, you associate a font transformation matrix with an `ATSUStyle` object. You can set the values in the font transformation matrix to achieve such effects as reversing glyphs across the X-axes and rotating glyphs Note that ATSUI's layout uses the transformed metrics so layout will be effected and in some cases the effects might be unexpected. For example, for a transformation that mirrors the glyph across the Y-axes the metrics are in reverse and glyphs are rendered on top of each other.

Declared in `ATSUnicodeTypes.h`.

Available starting with Mac OS X version 10.2.

`kATSUStyleUnderlineCountOptionTag`

Specifies the number of strokes to be drawn for an underline. The associated value is of type `ATSUStyleLineCountType`. The default value is `kATSUStyleSingleLineCount`. May be set as a style attribute.

Available in Mac OS X version 10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStyleUnderlineColorOptionTag`

Specifies the color of the strokes to draw for an underlined run of text. The associated value is of type `CGColorRef`. The default value is `NULL`. If `NULL`, the text color is used. The CGColor object (`CGColorRef`) is retained by the style object in which it is set. May be set as a style attribute.

Declared in `ATSUnicodeTypes.h`.

Available in Mac OS X version 10.3 and later.

`kATSUStyleStrikeThroughTag`

Specifies strikethrough style. The associated value is of type `Boolean`. The default value is `false`. May be set as a style attribute.

Available in Mac OS X version 10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStyleStrikeThroughCountOptionTag`

Specifies the number of strokes to be drawn for a strikethrough. The associated value is of type `ATSUStyleLineCountType`. The default value is `kATSUStyleSingleLineCount`. May be set as a style attribute.

Available in Mac OS X version 10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStyleStrikeThroughColorOptionTag`

Specifies the color of the strokes to draw for a strikethrough style. The associated value is of type `CGColorRef`. The CGColor object (`CGColorRef`) is retained by the style object in which it is set. The default value is `NULL`. If `NULL`, the text color is used. May be set as a style attribute.

Declared in `ATSUnicodeTypes.h`.

Available in Mac OS X version 10.3 and later.

`kATSUStyleDropShadowTag`
Specifies the text should be drawn with a drop shadow. The associated value is of type `Boolean`. The default value is `false`. Only takes effect if a CGContext is used for drawing. If you set this style attribute, you also need to set the drop shadow color using the tag `kATSUStyleDropShadowColorOptionTag`.

Declared in `ATSUnicodeTypes.h`.

Available in Mac OS X version 10.3 and later.

`kATSUStyleDropShadowBlurOptionTag`
Specifies the amount of blur for a drop shadow. The associated value is of type `float`. The default value is `0.0`. May be set as a style attribute.

Available in Mac OS X version 10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStyleDropShadowColorOptionTag`
Specifies the color and opacity of a drop shadow. The associated value is of type `CGColorRef`. The default value is `NULL`. You need to set the `CGColorRef` to a value other than `NULL` if you want to see the drop shadow. May be set as a style attribute.

Available in Mac OS X version 10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStyleDropShadowOffsetOptionTag`
Specifies the amount of offset from the text to be used when drawing a drop shadow. The associated value is of type `CGSize`. The default value is (`3.0`, `-3.0`). May be set as a style attribute.

Available in Mac OS X version 10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUMaxStyleTag`
A convenience tag that specifies the upper limit of style attribute tags.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULanguageTag`
This tag is obsolete. Instead use `kATSULangRegionTag`.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUMaxATSUITagValue`
Specifies this maximum Apple ATSUI reserved tag value. If you define a tag, it must have a value larger than the value of this tag.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion
An attribute tag cannot be used in versions of the Mac OS that are earlier than the version in which the tag was introduced. For example, a tag available in Mac OS version 10.2 cannot be used in Mac OS version 10.1 or earlier. You can call the function `Gestalt` to check version information for ATSUI.

Attribute tags indicates the particular type of attribute under consideration: font, size, color, and so on. Each style run may have at most one attribute with a given attribute tag (that is, a style run can't have more than one font or size) but may have none.

Some of the constants specify attributes that are applied to a style run, while other attributes are applied to an entire text layout object or to just a line in a text layout object. The constant descriptions assume horizontal text. If you set or get the an attribute that has been set for vertical text, you should interpret the constant descriptions accordingly.

Most of the constants in this section are described in further detail in *Inside Mac OS X: Rendering Unicode Text With ATSUI*. Where appropriate, that document provides illustrations that show the effect of applying an attribute. It also describes how to write code that sets style, line, and layout attributes.

A style run may have at most one style attribute with a given attribute tag. That is, a style run can't have more than one font or size attribute set but the style run does not need to have any attribute set explicitly.

When you set an attribute value for a line, the value overrides the attribute value set for the text layout object that contains the line. This is true even if you set line attributes before you set attributes for the entire text layout object that contains the line.

You can create your own attribute tag as long as your tag is outside those values reserved by Apple— 0 to 65,535 (0 to 0x0000FFFF). See *Rendering Unicode Text With ATSUI* for information on creating and registering your own attribute tags.

Background Data Types

Specify the data type of the background—a color or a callback.

```
typedef UInt32 ATSUBackgroundDataType;
enum {
    kATSUBackgroundColor = 0,
    kATSUBackgroundCallback = 1
};
```

Constants

`kATSUBackgroundColor`

Specifies the data type of the text background is a color.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUBackgroundCallback`

Specifies the data type of the text background is a callback.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Caret Movement Types

Specify the unit distance by which the caret moves.

```
typedef UInt16 ATSUCursorMovementType;
enum {
    kATSUByCharacter = 0,
    kATSUByTypographicCluster = 1,
    kATSUByWord = 2,
    kATSUByCharacterCluster = 3,
    kATSUByCluster = 1
};
```

Constants

`kATSUByCharacter`
Specifies to move the caret by a units based on single characters.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUByTypographicCluster`
Specifies to move the caret by units of clusters based on characters or ligatures.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUByWord`
Specifies to move the caret by units based on words.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUByCharacterCluster`
Specifies to move the caret by units based only on clusters of characters.
Available only in Mac OS X and in CarbonLib versions 1.3 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUByCluster`
An obsolete name for the constant `kATSUByTypographicCluster`.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

Discussion

A caret movement type is used to indicate the unit (character, word, and so on) by which to move the caret. You use these constants when you call the ATSUI caret movement functions. Functions that use caret movement types use this information to calculate the edge offset in memory that corresponds to the resulting cursor position.

Convenience Constants

Specify whether to clear values or whether drawing, measuring, or hit-testing should be done relative to the current pen location in the current graphics port.

```
enum {
    kATSUUseGrafPortPenLoc = (unsigned long)0xFFFFFFFF,
    kATSUClearAll = (unsigned long)0xFFFFFFFF
};
```

Constants

`kATSUUseGrafPortPenLoc`

Indicates that drawing, measuring, or hit-testing should be done relative to the current pen location in the current graphics port.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUClearAll`

Removes all previously set values from a style object, a single line, or a text layout object.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion

You can pass the `kATSUUseGrafPortPenLoc` constant to functions that operate on text layout objects to indicate that drawing, measuring, or hit-testing should be done relative to the current pen location in the current graphics port.

You can pass the `kATSUClearAll` constant to the following functions to remove previously set values from a style object: to `ATSUClearAttributes` (page 1848) to remove style run attributes, to `ATSUClearFontFeatures` (page 1849) to remove font features, and to `ATSUClearFontVariations` (page 1850) to remove font variations.

You can also use the `kATSUClearAll` constant to remove previously set text layout attributes: to `ATSUClearLineControls` (page 1853), to remove text layout attributes from a single line of a text layout object, and to `ATSUClearLayoutControls` (page 1852) to remove text layout attributes from every line in a text layout object.

Direct Data Selectors

Specify the layout data to obtain when calling the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout`.

```
typedef UInt32 ATSUDirectDataSelector;
enum {
    kATSUDirectDataAdvanceDeltaFixedArray = 0L,
    kATSUDirectDataBaselineDeltaFixedArray = 1L,
    kATSUDirectDataDeviceDeltaSInt16Array = 2L,
    kATSUDirectDataStyleIndexUInt16Array = 3L,
    kATSUDirectDataStyleSettingATSUStyleSettingRefArray = 4L,
    kATSUDirectDataLayoutRecordATSLayoutRecordVersion1 = 100L,
    kATSUDirectDataLayoutRecordATSLayoutRecordCurrent =
            kATSUDirectDataLayoutRecordATSLayoutRecordVersion1
};
```

Constants

`kATSUDirectDataAdvanceDeltaFixedArray`

Specifies the parallel advance delta (delta X) array, which is an array of `Fixed` values. This array is created only on demand. If you plan to modify the data in this array, you should set the `iCreate` parameter to `true` when you call the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) to obtain this array.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeDirectAccess.h`.

`kATSUDirectDataBaselineDeltaFixedArray`

Specifies the parallel baseline delta (delta Y) array, which is an array of `Fixed` values. This array is created only on demand. If you plan to modify the data in this array, you should set the `iCreate` parameter to `true` when you call the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) to obtain this array.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeDirectAccess.h`.

`kATSUDirectDataDeviceDeltaSInt16Array`

Specifies the parallel device delta array, which is an array of `SInt16` values used to adjust truncated fractional values for devices that do not accept fractional positioning. The array specified by this selector is also used to provide precise positioning for connected scripts. This array is created only on demand. If you plan to modify the data in this array, you should set the `iCreate` parameter to `true` when you call the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) to obtain this array.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeDirectAccess.h`.

`kATSUDirectDataStyleIndexUInt16Array`

Specifies the parallel style index array, which is an array of (`UInt16`) values. The values in this array are indexes into the style setting reference (`ATSUStyleSettingRef`) array. This array is created only on demand. If you plan to modify the data in this array, you should set the `iCreate` parameter to `true` when you call the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) to obtain this array.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeDirectAccess.h`.

`kATSUDirectDataStyleSettingATSUStyleSettingRefArray`
Specifies the style setting reference (`ATSUStyleSettingRef`) array. This array is always available if the text layout object has any text associated with it. Setting the `iCreate` parameter when you call the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) to obtain this array has no effect.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeDirectAccess.h`.

`kATSUDirectDataLayoutRecordATSLayoutRecordVersion1`
Specifies the `ATSLayoutRecord` array, with the version 1 of the `ATSLayoutRecord` data structure. You should not use this selector. Instead use the selector `kATSUDirectDataLayoutRecordATSLayoutRecordCurrent` to ensure that your code uses the most current version of the `ATSLayoutRecord` data structure. ATSUI performs the most efficient processing only for the latest version of `ATSLayoutRecord` data structure. This array is always available if the text layout object has any text associated with it. Setting the `iCreate` parameter when you call the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) to obtain this array has no effect.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeDirectAccess.h`.

`kATSUDirectDataLayoutRecordATSLayoutRecordCurrent`
Specifies the `ATSLayoutRecord` array, with the current version of the `ATSLayoutRecord` data structure. Always use this selector to get the array of `ATSLayoutRecord` data structures. This array is always available if the text layout object has any text associated with it. Setting the `iCreate` parameter when you call the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873) to obtain this array has no effect.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeDirectAccess.h`.

Discussion
You can provide direct data selectors to the functions `ATSUDirectGetLayoutDataArrayPtrFromLineRef` (page 1872) or `ATSUDirectGetLayoutDataArrayPtrFromTextLayout` (page 1873).

Flattened Data Font Type Selectors

Specifies the data type for flattened font name data.

```
typedef UInt32 ATSFlatDataFontSpeciferType;
enum {
    kATSFlattenedFontSpecifierRawNameData = 'namd'
};
```

Constants
`kATSFlattenedFontSpecifierRawNameData`
Specifies to use the font name as the flattened font name.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeFlattening.h`.

Flattened Data Format Selectors

Specify the format to use when flattening or unflattening data.

```
typedef UInt32 ATSUFlattenedDataStreamFormat;
enum {
    kATSUDataStreamUnicodeStyledText = 'ustl'
};
```

Constants

`kATSUDataStreamUnicodeStyledText`
Specifies to use the `'ustl'` data specification when flattening or unflattening data.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeFlattening.h`.

Flattened Style Run Data Options

Specify options to use when flattening ATSUI style run data.

```
typedef UInt32 ATSUFlattenStyleRunOptions;
enum {
        kATSUFlattenOptionNoOptionsMask = 0x00000000
};
```

Constants

`kATSUFlattenOptionNoOptionsMask`
Specifies that no options are to be used.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeFlattening.h`.

Discussion
Additional options may be added in the future.

Flattened Data Version Numbers

Specify versions of the `'ustl'` specification.

```
enum {
    kATSFlatDataUstlVersion0      = 0,
    kATSFlatDataUstlVersion1      = 1,
    kATSFlatDataUstlVersion2      = 2,
    kATSFlatDataUstlCurrentVersion = kATSFlatDataUstlVersion2};
```

Constants

`kATSFlatDataUstlVersion0`
Specifies version 0. This version is obsolete.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeFlattening.h`.

`kATSFlatDataUstlVersion1`
Specifies version 1. This version is obsolete.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeFlattening.h`.

`kATSFlatDataUstlVersion2`
Specifies version 2.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeFlattening.h`.

`kATSFlatDataUstlCurrentVersion`
Specifies the current version.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeFlattening.h`.

Discussion
The ATSUI functions `ATSUFlattenStyleRunsToStream` and `ATSUUnflattenStyleRunsFromStream` operate on data that conform to version 2 of the `'ustl'` specification.

Font Fallback Methods

Specify the method by which ATSUI tries to find an appropriate font for a character if the assigned font does not contain the needed glyphs.

```
typedef UInt16 ATSUFontFallbackMethod;
enum {
    kATSUDefaultFontFallbacks = 0,
    kATSULastResortOnlyFallback = 1,
    kATSUSequentialFallbacksPreferred = 2,
    kATSUSequentialFallbacksExclusive = 3
};
```

Constants
`kATSUDefaultFontFallbacks`
Specifies to use ATSUI's default font search method. ATSUI searches through all available fonts on the system for one that matches any text that cannot be drawn with the font specified in the current ATSU style object (`ATSUStyle`). ATSUI first searches in the standard application fonts for various languages. If that fails, it searches through the remaining fonts on the system in whatever order the Font Manager returns them. After ATSUI has searched all the fonts in the system, any unmatched text is drawn with the last-resort font.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSULastResortOnlyFallback`
Specifies that ATSUI should use the last resort font if the assigned font does not contain the needed glyphs.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUSequentialFallbacksPreferred`
Specifies that ATSUI should first search sequentially through the list of supplied fonts before it searching through all available fonts on the system.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUSequentialFallbacksExclusive`
Specifies that ATSUI should search exclusively through the list of supplied fonts. ATSUI use the last-resort font if it does not find a match in the list of supplied fonts.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Glyph Origin Selectors

Specify which glyph origin to use to determine the width of the typographic glyph bounds.

```
enum {
    kATSUseCaretOrigins           = 0,
    kATSUseDeviceOrigins          = 1,
    kATSUseFractionalOrigins      = 2,
    kATSUseOriginFlags            = 3
};
```

Constants

`kATSUseCaretOrigins`
Specifies to use the caret origin to determine the width of the typographic glyph bounds. The caret origin is halfway between two characters.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSUseDeviceOrigins`
Specifies to use the glyph origin in device space to determine the width of the typographic glyph bounds. This is useful if you need to adjust text on the screen.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSUseFractionalOrigins`
Specifies to use the glyph origin in fractional absolute positions (which are uncorrected for display device) to determine the width of the typographic glyph bounds. This provides the ideal position of laid-out text and is useful if you need to scale text on the screen. The glyph origin is also used to obtain the width of the typographic bounding rectangle when you call the function `ATSUMeasureText`.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSUseOriginFlags`
The number of glyph origin selectors.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

Discussion
You can pass a glyph bounds selector in the `iTypeOfBounds` parameter of the function `ATSUGetGlyphBounds` (page 1904) to indicate whether the width of the resulting typographic glyph bounds is determined using the caret origin, glyph origin in device space, or glyph origin in fractional absolute positions.

Glyph Collection Types

Specify a character set.

```
typedef UInt16 GlyphCollection;
enum {
    kGlyphCollectionGID           = 0,
    kGlyphCollectionAdobeCNS1     = 1,
    kGlyphCollectionAdobeGB1      = 2,
    kGlyphCollectionAdobeJapan1   = 3,
    kGlyphCollectionAdobeJapan2   = 4,
    kGlyphCollectionAdobeKorea1   = 5,
    kGlyphCollectionUnspecified   = 0xFF
};
```

Constants

`kGlyphCollectionGID`
Indicates that the glyph value represents the actual glyph ID of a specific font.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kGlyphCollectionAdobeCNS1`
Specifies Adobe CNS1 CID-keyed fonts.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kGlyphCollectionAdobeGB1`
Specifies Adobe GB1 CID-keyed fonts.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kGlyphCollectionAdobeJapan1`
Specifies Adobe Japan1 CID-keyed fonts.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kGlyphCollectionAdobeJapan2`
Specifies Adobe Japan2 CID-keyed fonts.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kGlyphCollectionAdobeKorea1`
Specifies Adobe Korea1 CID-keyed fonts.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kGlyphCollectionUnspecified`
Indicates that the glyph collection is not specified.

Available in Mac OS X v10.2 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion
A CID-keyed font is a PostScript font that uses a font file format developed by Adobe for fonts that have large character sets, such as Chinese, Japanese, and Korean fonts. For more information on CID-keyed fonts, see the Adobe website:

http://partners.adobe.com/

Glyph Direction Selectors

Specify a glyph direction.

```
enum {
    kATSULeftToRightBaseDirection = 0,
    kATSURightToLeftBaseDirection = 1
};
```

Constants
`kATSULeftToRightBaseDirection`
Imposes left-to-right direction on glyphs in a line of horizontal text; for vertical text, imposes top-to-bottom direction.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSURightToLeftBaseDirection`
Imposes right-to-left direction on glyphs in a line of horizontal text; for vertical text, imposes bottom-to-top direction.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion
These constants specify values for the `kATSULineDirectionTag` attribute tag. You can use one of these constants to set or obtain glyph direction in a line of text or an entire text layout object, regardless of their font-specified direction; see the functions `ATSUSetLayoutControls` (page 1955), `ATSUSetLineControls` (page 1956), `ATSUGetLayoutControl` (page 1912), and `ATSUGetLineControl` (page 1913).

Glyph Property Flags

Specify properties for a glyph.

```
typedef UInt32 ATSGlyphInfoFlags;
enum {
    kATSGlyphInfoAppleReserved    = 0x1FFBFFE8,
    kATSGlyphInfoIsAttachment     = (unsigned long)0x80000000,
    kATSGlyphInfoIsLTHanger       = 0x40000000,
    kATSGlyphInfoIsRBHanger       = 0x20000000,
    kATSGlyphInfoTerminatorGlyph  = 0x00080000,
    kATSGlyphInfoIsWhiteSpace     = 0x00040000,
    kATSGlyphInfoHasImposedWidth  = 0x00000010,
    kATSGlyphInfoByteSizeMask     = 0x00000007
};
```

Constants

`kATSGlyphInfoAppleReserved`
This flag is reserved by Apple. If you try to use it you may get an invalid value error.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSGlyphInfoIsAttachment`
Specifies that the glyph attaches to another glyph.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSGlyphInfoIsLTHanger`
Specifies that the glyph can hang off the left or top edge of a line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSGlyphInfoIsRBHanger`
Specifies that the glyph can hang off the right or bottom edge of a line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSGlyphInfoTerminatorGlyph`
Specifies that the glyph is not truly a glyph, but an end-marker to allow the calculation of the previous glyph's advance.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSGlyphInfoIsWhiteSpace`
Specifies that the glyph is a whitespace glyph.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSGlyphInfoHasImposedWidth`
Specifies that the glyph has an imposed width (that is, an advance width) specified by the style.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSGlyphInfoByteSizeMask`
Specifies the size of the character that spawned the glyph. This is a three-bit mask that you can use to obtain the size of the original character that spawned a glyph. If you perform a logical `and` operation between this mask and an `ATSGlyphInfoFlags` flag, you obtain the size in bytes of the original character (0 - 7 bytes).

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

Discussion
Glyph information flags are set in the individual `ATSLayoutRecord` structure and apply only to the `ATSGlyphRef` reference in that structure. The flags are used by the ATSUI to tag a glyph with one or more specific properties.

Highlight Methods

Specify a text highlighting method.

```
typedef UInt32 ATSUHighlightMethod;
enum {
    kInvertHighlighting = 0,
    kRedrawHighlighting = 1
};
```

Constants
`kInvertHighlighting`
Specifies to use inversion for highlighting. You can use this when the background is a single color.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kRedrawHighlighting`
Specifies to use your callback for highlighting. You should use this when the background is complex (containing, for example, multiple colors, patterns, or pictures).

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion
You set the highlighting method by calling the function `ATSUSetHighlightingMethod` (page 1953).

Invalid Font ID Constant

Specifies a Font ID is not valid.

```
enum {
    kATSUInvalidFontID = 0
};
```

Constants
`kATSUInvalidFontID`
Indicates that the font ID is invalid.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion
The functions `ATSUFONDtoFontID` (page 1884), `ATSUFindFontFromName` (page 1879), and `ATSUMatchFontsToText` (page 1936) pass back this constant to indicate an invalid font ID. This constant is available with ATSUI 1.0.

Line Truncation Selectors

Specify where in a line truncation should occur.

```
typedef UInt32 ATSULineTruncation;
enum {
    kATSUTruncateNone = 0,
    kATSUTruncateStart = 1,
    kATSUTruncateEnd = 2,
    kATSUTruncateMiddle = 3,
    kATSUTruncateSpecificationMask = 7,
    kATSUTruncFeatNoSquishing = 8
};
```

Constants

`kATSUTruncateNone`
Specifies not to truncate the line.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUTruncateStart`
Specifies to truncate the line at the beginning.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUTruncateEnd`
Specifies to truncate the line at the end.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUTruncateMiddle`
Specifies to truncate the line in the middle
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUTruncateSpecificationMask`
Reserved for the truncation specification (0 - 7).
Available in Mac OS X v10.1 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUTruncFeatNoSquishing`
Specifies not to perform any negative justification in lieu of truncation.
Available in Mac OS X v10.1 and later.
Declared in `ATSUnicodeTypes.h`.

Discussion
Line truncation options specify values for the `kATSULineTruncation` attribute tag. You can add any line truncation option to the option `kATSUTruncateSpecificationMask`. For example, adding `kATSUTruncateEnd` and `kATSUTruncFeatNoSquishing` to the mask `kATSUTruncateSpecificationMask` results in the value `0x0000000A`.

Layout Callback Status Values

Specify the status of a layout operation override callback.

```
typedef UInt32 ATSULayoutOperationCallbackStatus;
enum {
    kATSULayoutOperationCallbackStatusHandled = 0x00000000,
    kATSULayoutOperationCallbackStatusContinue = 0x00000001
};
```

Constants

`kATSULayoutOperationCallbackStatusHandled`
Specifies that your callback function has handled the operation which triggered the callback. This indicates to ATSUI that it does not need to perform any further processing for the layout operation.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationCallbackStatusContinue`
Specifies that your callback function has not handled the operation which triggered the callback. This indicates to ATSUI that needs to perform its own processing for the layout operation.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

Discussion
You must return one of these status values from your `ATSUDirectLayoutOperationOverrideProcPtr` (page 1998) callback function to indicate to ATSUI whether or not your callback handled the layout operation.

Layout Operation Selectors

Specify a layout operation.

```
typedef UInt32 ATSULayoutOperationSelector;
enum {
    kATSULayoutOperationNone      = 0x00000000,
    kATSULayoutOperationJustification = 0x00000001,
    kATSULayoutOperationMorph     = 0x00000002,
    kATSULayoutOperationKerningAdjustment = 0x00000004,
    kATSULayoutOperationBaselineAdjustment = 0x00000008,
    kATSULayoutOperationTrackingAdjustment = 0x00000010,
    kATSULayoutOperationPostLayoutAdjustment = 0x00000020,
    kATSULayoutOperationAppleReserved = (unsigned long)0xFFFFFFC0
};
```

Constants

`kATSULayoutOperationNone`
Specifies that no layout operation is currently selected.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationJustification`
Specifies the justification operation.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationMorph`
Specifies the character-morphing operation.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationKerningAdjustment`
Specifies the kerning-adjustment operation.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationBaselineAdjustment`
Specifies the baseline-adjustment operation.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationTrackingAdjustment`
Specifies the tracking-adjustment operation.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationPostLayoutAdjustment`
Specifies the period of time after ATSUI has completed its layout operations.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

`kATSULayoutOperationAppleReserved`
This selector is reserved for future use.
Available in Mac OS X v10.2 and later.
Declared in `ATSLayoutTypes.h`.

Discussion
You can use layout operation selectors to specify to ATSUI which operations to override. These selectors can also be passed from ATSUI to your application to indicate which operation is currently in progress.

Line Alignment Selectors

Specify the alignment of text relative to the margins in a line of text or in an entire text layout object.

```
#define kATSUStartAlignment ((Fract) 0x00000000L)
#define kATSUEndAlignment ((Fract) 0x40000000L)
#define kATSUCenterAlignment ((Fract) 0x20000000L)
```

Constants

`kATSUStartAlignment`
Specifies that horizontal text should be drawn to the right of the left margin (that is, its left edge coincides with the text layout object's position plus text width). Vertical text should be drawn below the top margin.

`kATSUEndAlignment`
Specifies that horizontal text should be drawn to the left of the right margin. Vertical text should be drawn above the bottom margin.

`kATSUCenterAlignment`
Specifies that horizontal text should be drawn between the left and right margins with an equal amount of space on either side. Vertical text should be drawn between the top and bottom margins with an equal amount of space on either side.

Discussion
You can use one of these constants to set or obtain the alignment of text relative to the margins in a line of text or in an entire text layout object; see the functions `ATSUSetLayoutControls` (page 1955), `ATSUSetLineControls` (page 1956), `ATSUGetLayoutControl` (page 1912), and `ATSUGetLineControl` (page 1913), respectively.

Line Height and Font Tracking Selectors

Specify how to determine line height and whether to turn off font tracking.

```
enum {
    kATSUseGlyphAdvance = 0x7FFFFFFF,
    kATSUseLineHeight = 0x7FFFFFFF,
    kATSNoTracking = (long)0x80000000
};
```

Constants

`kATSUseGlyphAdvance`
Specifies that ATSUI use the natural glyph advance value in a line or entire text layout object.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSUseLineHeight`
Specifies that ATSUI use the natural line ascent and descent values dictated by the font and pixel size to determine line ascent and descent in a line or entire text layout object.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSNoTracking`
A value of type `negativeInfinity` that indicates that font tracking should be off.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

Discussion
You use line height selectors to set line ascent and descent text layout attributes. You can set the line ascent text layout attribute for a line or an entire text layout object by passing the `kATSULineAscentTag` tag to the functions `ATSUSetLineControls` (page 1956) and `ATSUSetLayoutControls` (page 1955), respectively. You can set the line descent text layout attribute for a line or an entire text layout object by passing the `kATSULineDescentTag` tag to the functions `ATSUSetLineControls` (page 1956) and `ATSUSetLayoutControls` (page 1955), respectively.

Line Justification Selectors

Specify the degree of line justification for a single line or an entire text layout object.

```
#define kATSUNoJustification        ((Fract) 0x00000000L)
#define kATSUFullJustification      ((Fract) 0x40000000L)
```

Constants
`kATSUNoJustification`
Indicates no justification.

`kATSUFullJustification`
Full justification between the text margins. White space is “stretched” to make the line extend to both text margins.

Discussion
You can set the line justification text layout attribute for a line or an entire text layout object by passing the `kATSULineJustificationFactorTag` tag the functions `ATSUSetLineControls` (page 1956) and `ATSUSetLayoutControls` (page 1955), respectively.

Line Layout Attribute Tags

Specify line layout attributes to be applied at the line level.

```
typedef UInt32 ATSLineLayoutOptions;
enum {
    kATSLineNoLayoutOptions       = 0x00000000,
    kATSLineIsDisplayOnly         = 0x00000001,
    kATSLineHasNoHangers          = 0x00000002,
    kATSLineHasNoOpticalAlignment = 0x00000004,
    kATSLineKeepSpacesOutOfMargin = 0x00000008,
    kATSLineNoSpecialJustification = 0x00000010,
    kATSLineLastNoJustification   = 0x00000020,
    kATSLineFractDisable          = 0x00000040,
    kATSLineImposeNoAngleForEnds  = 0x00000080,
    kATSLineFillOutToWidth        = 0x00000100,
    kATSLineTabAdjustEnabled      = 0x00000200,
    kATSLineIgnoreFontLeading     = 0x00000400,
    kATSLineApplyAntiAliasing     = 0x00000800,
    kATSLineNoAntiAliasing        = 0x00001000,
    kATSLineDisableNegativeJustification = 0x00002000,
    kATSLineDisableAutoAdjustDisplayPos = 0x00004000,
    kATSLineUseQDRendering        = 0x00008000,
    kATSLineDisableAllJustification = 0x00010000,
    kATSLineDisableAllGlyphMorphing = 0x00020000,
    kATSLineDisableAllKerningAdjustments = 0x00040000,
    kATSLineDisableAllBaselineAdjustments = 0x00080000,
    kATSLineDisableAllTrackingAdjustments = 0x00100000,
    kATSLineDisableAllLayoutOperations = kATSLineDisableAllJustification
|
        kATSLineDisableAllGlyphMorphing |
        kATSLineDisableAllKerningAdjustments |
        kATSLineDisableAllBaselineAdjustments |
        kATSLineDisableAllTrackingAdjustments,
    kATSLineUseDeviceMetrics      = 0x01000000,
    kATSLineBreakToNearestCharacter = 0x02000000,
    kATSLineAppleReserved         = (unsigned long)0xFCE00000};
```

Constants

`kATSLineNoLayoutOptions`

Specifies not to apply any options.

Available i n ATSUI 1.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineIsDisplayOnly`

This line option is no longer used. Instead use `kATSLineUseDeviceMetrics`.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineHasNoHangers`

Specifies not to form hanging punctuation on the line. If the bit specified by this mask is set, the automatic hanging punctuation in the text layout object is overridden. The value in this bit overrides any adjustment to hanging punctuation set for a style run inside the text layout object using the style run attribute tags `kATSUForceHangingTag` or `kATSUHangingInhibitFactorTag`.

Declared in `ATSLayoutTypes.h`.

Available in ATSUI 1.0 and later.

`kATSLineHasNoOpticalAlignment`
Specifies not to perform optical alignment on the line. Optical alignment adjusts characters at the text margin so that they appear to be properly aligned; strict alignment can often cause the illusion of a ragged edge. The value in this bit overrides any adjustment to optical alignment set for a style run inside the text layout object using the style run attribute tag `kATSUNoOpticalAlignmentTag`.

Declared in `ATSLayoutTypes.h`.

Available in ATSUI 1.0 and later.

`kATSLineKeepSpacesOutOfMargin`
Specifies that the trailing white spaces at the end of a line of justified text should be placed outside the margin.

Available in ATSUI 1.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineNoSpecialJustification`
Specifies not to perform post-compensation justification on the line, even if such processing is necessary. This flag cannot be set for a single line of a text layout object. The value in this bit overrides any adjustment to the postcompensation actions set for a style run using the style run attribute tag `kATSUNoSpecialJustificationTag`.

Declared in `ATSLayoutTypes.h`.

Available in ATSUI 1.0 and later.

`kATSLineLastNoJustification`
Specifies not to justify a line if it is the last line of a justified text layout object. This flag is meaningless when setting a line's text layout attributes.

Available in ATSUI 1.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineFractDisable`
Specifies to position of the text in the line or text layout object relative to fractional absolute positions, which are uncorrected for device display. This provides the ideal position of laid-out text and is useful for scaling text onscreen. This origin is also used to get the width of the typographic bounding rectangle when you call the function `ATSUGetUnjustifiedBounds` (page 1923).

Declared in `ATSLayoutTypes.h`.

Available in ATSUI 1.1 and later.

`kATSLineImposeNoAngleForEnds`
Specifies to draw the carets on the far right and left sides of an unrotated line as vertical, no matter what the angle of text.

Available in ATSUI 1.1 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineFillOutToWidth`
Specifies to extend highlighting to both ends of a line, regardless of caret locations. This option does not effect the caret locations. This is provided for your convenience to extend your highlighting to the full width of the line.

Available in ATSUI 1.1 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineTabAdjustEnabled`
Specifies to automatically adjust the tab character width so that it fits the specified line width. If you are using ATSUI's tab functions—`ATSUSetTabArray` (page 1962) and `ATSUGetTabArray` (page 1918) to define a tab rule you do not need to use this selector. The selector is useful if you are handling your own tabs and only applies if the tab is at the end of a line (backing store). You must set this bit to ensure that highlighting is done correctly across tab stops. To ensure this, you should also set the bit specified by the `kATSLineImposeNoAngleForEnds` mask constant.

Declared in `ATSLayoutTypes.h`.

Available in ATSUI 1.2 and later.

`kATSLineIgnoreFontLeading`
Specifies to ignore any leading value specified by a font.

Available in ATSUI 2.3 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineApplyAntiAliasing`
Specifies that Apple Type Services should produce antialiased glyph images even if system preferences or Quartz settings indicate otherwise.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineNoAntiAliasing`
Specifies that Apple Type Services should turn-off antialiasing glyph imaging even if system preferences or Quartz settings indicate otherwise. This option negates the `kATSLineApplyAntiAliasing` bit if it is set.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableNegativeJustification`
Specifies to allow glyph positions to extend beyond the line's assigned width if the line width is not sufficient to hold all its glyphs. This ensures that negative justification is not used.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableAutoAdjustDisplayPos`
Specifies not to automatically adjust individual character positions when rendering lines that have any integer glyph positioning, whether the integer glyph positioning is due to non-antialiased characters or though the use of the selector `kATSLineFractDisable`.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineUseQDRendering`
Specifies to use QuickDraw to render a line of text instead of the default ATSUI rendering. With Mac OS X version 10.2, ATSUI renders text through Quartz, even if you do not attach a `CGContext` to a text layout object. In the default case, ATSUI retrieves the internal canonical `CGContext` of the current port, and renders to that port using Quartz at an antialiasing setting that simulates QuickDraw rendering. That is, a 4-bit pixel-aligned antialiasing. Because the default setting gives you simulated QuickDraw rendering, you should use the tag `kATSLineUseQDRendering` only if you must have backward compatibility. With Mac OS X version 10.3, this option no longer does anything different from not declaring a `CGContext`.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableAllJustification`
Specifies not to perform any justification operations on the line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableAllGlyphMorphing`
Specifies not to perform any glyph-morphing operations on the line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableAllKerningAdjustments`
Specifies not to perform any kerning-adjustment operations on the line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableAllBaselineAdjustments`
Specifies not to perform any baseline-adjustment operations on the line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableAllTrackingAdjustments`
Specifies not to perform any tracking-adjustment operations on the line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineDisableAllLayoutOperations`
Specifies to turn off all layout adjustments for this line.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineUseDeviceMetrics`
Specifies to used rounded device metrics instead of fractional path metrics. This optimizes display of text and should be used only in cases in which the text is displayed onscreen as opposed to printed or output to PDF. If you use this option to display text onscreen as well as to print or create a PDF, you will get different results between the two types of output. This attribute is not recommended for Quartz antialiased text.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineBreakToNearestCharacter`
Specifies that line breaking should occur at the nearest character, not word. This could cause a word to be split over multiple lines.

Available in Mac OS X version 10.3 and later.

Declared in `ATSLayoutTypes.h`.

`kATSLineAppleReserved`
This selector is reserved by Apple. If you try to use it, ATSUI returns the `kATSUInvalidAttributeValueEr` result code.

Available in ATSUI 1.1 and later.

Declared in `ATSLayoutTypes.h`.

Discussion
You can use a constant of type `ATSLineLayoutOptions` to set or obtain the line layout options in a line of text or for an entire text layout object; see the functions `ATSUSetLineControls` (page 1956) and `ATSUSetLayoutControls` (page 1955), respectively.

Line Layout Width Selector

Specifies a line width.

```
enum {
    kATSUUseLineControlWidth = 0X7FFFFFFF
};
```

Constants
`kATSUUseLineControlWidth`
Indicates that the functions `ATSUBreakLine` or `ATSUBatchBreakLines` should use the previously set line width attribute for the current line to determine how many characters can fit on the line. If no line width has been set for the line, these functions use the line width set for the text layout object; if not set, these functions use the default line width value.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion
You can pass this constant to the functions `ATSUBreakLine` (page 1846) or `ATSUBatchBreakLines` (page 1844) to indicate that the function should use the line width previously set for that line to calculate the soft line break. If no line width has been set for the line, these functions use the line width set for the text layout object.

No Selectors Option

Specifies no selectors are chosen.

```
enum {
    kATSUNoSelector = 0x0000FFFF
};
```

Constants
`kATSUNoSelector`
Specifies no selectors are chosen.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Style Comparison Options

Specify how two style objects compare to each other.

```
typedef UInt16 ATSUStyleComparison;
enum {
    kATSUStyleUnequal = 0,
    kATSUStyleContains = 1,
    kATSUStyleEquals = 2,
    kATSUStyleContainedBy = 3
};
```

Constants

`kATSUStyleUnequal`
Specifies that styles are unequal.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUStyleContains`
Specifies that style 1 contains style 2 as a proper subset.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUStyleEquals`
Specifies that style 1 equals style 2.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUStyleContainedBy`
Specifies that style 1 is contained by style 2.
Available in Mac OS X v10.0 and later.
Declared in `ATSUnicodeTypes.h`.

Discussion
The function `ATSUCompareStyles` (page 1855) returns a constant of type `ATSUStyleComparison` to indicate whether two style objects are the same, different, or a subset of one another.

Style Line Count Types

Specifies how many lines to draw for a given style type.

```
typedef UInt16 ATSUStyleLineCountType;
enum {
    kATSUStyleSingleLineCount     = 1,
    kATSUStyleDoubleLineCount     = 2
};
```

Constants

`kATSUStyleSingleLineCount`
Specifies to use a single line.
Available in Mac OS X v10.3 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUStyleDoubleLineCount`
Specifies to use a double line.
Available in Mac OS X v10.3 and later.
Declared in `ATSUnicodeTypes.h`.

Discussion
These constants are available in Mac OS X version 10.3 and later. Currently only the underline and strike through styles support this type.

Style Rendering Options

Specify rendering options for a style object.

```
typedef UInt32 ATSStyleRenderingOptions;
enum {
    kATSStyleNoOptions            = 0x00000000,
    kATSStyleNoHinting            = 0x00000001,
    kATSStyleApplyAntiAliasing    = 0x00000002,
    kATSStyleNoAntiAliasing       = 0x00000004,
    kATSStyleAppleReserved        = (unsigned long)0xFFFFFFF8,
    kATSStyleApplyHints           = kATSStyleNoOptions
};
```

Constants

`kATSStyleNoOptions`
Specifies no options are set.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSStyleNoHinting`
Specifies that Apple Type Services (ATS) should produce unhinted glyph outlines. The default behavior is for ATS to produce is hinted glyph outlines.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSStyleApplyAntiAliasing`
Specifies that Apple Type Services should produce antialiased glyph images even if system preferences or Quartz 2D settings indicate otherwise.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSStyleNoAntiAliasing`
Specifies that Apple Type Services should turn-off antialiasing glyph imaging even if system preferences or Quartz 2D settings indicate otherwise. This selector negates the `kATSStyleApplyAntiAliasing` selector if is set.

Available in Mac OS X v10.2 and later.

Declared in `ATSLayoutTypes.h`.

`kATSStyleAppleReserved`
This selector is reserved by Apple. If you try to use it, you will get an invalid value error.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

`kATSStyleApplyHints`
Specifies that Apple Type Services should produce hinted glyph outlines. This selector is obsolete; do not use it. It is listed here only for backwards compatibility.

Available in Mac OS X v10.0 and later.

Declared in `ATSLayoutTypes.h`.

Discussion
You can set style rendering attributes for a style object (`ATSUStyle`) by using the `kATSUStyleRenderingOptionsTag` attribute tag and calling the function `ATSUSetAttributes` (page 1950). Style rendering options provide fine control over how a style is rendered.

Tab Positioning Options

Specify text positioning for ATSUI tab stops.

```
typedef UInt16 ATSUTabType;
enum {
    kATSULeftTab                    = 0,
    kATSUCenterTab                  = 1,
    kATSURightTab                   = 2,
    kATSUDecimalTab                 = 3,
    kATSUNumberTabTypes             = 4
};
```

Constants

`kATSULeftTab`
Specifies that the left side of the tabbed text should be flush against the tab stop.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUCenterTab`
Specifies that the tabbed text should be centered on the tab stop.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSURightTab`
Specifies that the right side of the tabbed text should be flush against the tab stop.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

`kATSUDecimalTab`
Specifies that the decimal point of a value should be centered on the tab stop. To set a decimal tab, use the tag `kATSULineDecimalTabCharacterTag`. This tag specifies the current setting for the decimal separator, and affects the behavior of decimal tabs for a text layout (not an individual line). The default character that is used as a separator is set by the user in System Preferences.
Declared in `ATSUnicodeTypes.h`.
Available in Mac OS X version 10.3 and later.

`kATSUNumberTabTypes`
Specifies the number of valid tab types.
Available in Mac OS X v10.2 and later.
Declared in `ATSUnicodeTypes.h`.

Discussion
You can use tab type constants to set a tab ruler. The default value is the user setting in System Preferences.

Text Buffer Convenience Constants

Refer to the beginning or end of a text buffer.

```
enum {
    kATSUFromTextBeginning = (unsigned long)0xFFFFFFFF,
    kATSUToTextEnd = (unsigned long)0xFFFFFFFF,
    kATSUFromPreviousLayout = (unsigned long)0xFFFFFFFE,
    kATSUFromFollowingLayout      = (unsigned long)0xFFFFFFFD
};
```

Constants

`kATSUFromTextBeginning`

Indicates that the range of text to be operated on should start at the beginning of the text layout object's text buffer.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUToTextEnd`

Indicates that the range of text to be operated on should span to the end of the text layout object's text buffer.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUFromPreviousLayout`

Used for bidirectional cursor movement between paragraphs in the functions `ATSURightwardCursorPosition` and `ATSULeftwardCursorPosition`.

Available in Mac OS X v10.3 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUFromFollowingLayout`

Used for bidirectional cursor movement between paragraphs in the functions `ATSURightwardCursorPostion` and `ATSULeftwardCursorPosition`.

Available in Mac OS X v10.3 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion

Do not pass these constants to functions which do not explicitly state they will accept them.

ATSUI functions that draw, highlight, measure, or otherwise operate on text do so to a range of text, not the entire text buffer (unless you specify the entire buffer). You specify the beginning of this range with an edge offset of type `UniCharArrayOffset`, and demarcate the end of the range by indicating a length of type `UniCharCount`.

If you want the range to start at the beginning of the text buffer, you should pass the constant `kATSUFromTextBeginning`. If you want the range to span the end of the text buffer, you should pass the constant `kATSUToTextEnd`. If you want the range to span the entire text buffer, pass `kATSUFromTextBeginning` in conjunction with the constant `kATSUToTextEnd`. For bidirectional text, you can specify the previous layout by passing the constant `kATSUFromPreviousLayout` and the following layout by passing the constant `kATSUFromFollowingLayout`.

Unflattened Style Run Data Options

Specify options to use when unflattening ATSUI style run data.

```
typedef UInt32 ATSUUnflattenStyleRunOptions;
enum {
    kATSUUnflattenOptionNoOptionsMask = 0x00000000
};
```

Constants
`kATSUUnflattenOptionNoOptionsMask`
Specifies that no options are to be used.

Discussion
Additional options may be added in the future.

Vertical Character Types

Specify the glyph orientation of font tracking settings or a style run.

```
typedef UInt16 ATSUVerticalCharacterType;
enum {
    kATSUStronglyHorizontal = 0,
    kATSUStronglyVertical = 1
};
```

Constants
`kATSUStronglyHorizontal`
Specifies a horizontal orientation.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

`kATSUStronglyVertical`
Specifies a vertical orientation.

Available in Mac OS X v10.0 and later.

Declared in `ATSUnicodeTypes.h`.

Discussion
You can pass a constant of type `ATSUVerticalCharacterType` to the functions `ATSUCountFontTracking` (page 1861) and `ATSUGetIndFontTracking` (page 1910) to specify the glyph orientation of font tracking settings, since font tracking settings differ depending upon glyph orientations.

You can also use one of these constants to set or obtain the glyph orientation of a style run; see the functions `ATSUSetAttributes` (page 1950) and `ATSUGetAttribute` (page 1892), respectively.

Result Codes

The most common result codes returned by Apple Type Services for Unicode Imaging are listed below.

Result Code	Value	Description
`kATSUInvalidTextLayoutErr`	-8790	The ATSUI text layout object is not initialized or is in an otherwise invalid state. Available beginning with ATSUI 1.0.

Result Code	Value	Description
kATSUInvalidStyleErr	-8791	The ATSUI style object is not initialized or in an otherwise invalid state. Available beginning with ATSUI 1.0.
kATSUInvalidTextRangeErr	-8792	The text range extends beyond the limits of the text layout object's text range. Available beginning with ATSUI 1.0.
kATSUFontsMatched	-8793	One or more characters cannot be rendered with the assigned font, but can be rendered with a substitute font from among those currently active. Available beginning with ATSUI 1.0.
kATSUFontsNotMatched	-8794	One or more characters can neither be rendered with the assigned font nor with any other currently active font. Available beginning with ATSUI 1.0.
kATSUNoCorrespondingFontErr	-8795	The font ID corresponds to an existing font that isn't available to ATSUI. Available beginning with ATSUI 1.0.
kATSUInvalidFontErr	-8796	The font ID does not correspond to any installed font or ATSUI is unable to obtain the font data (as in the case of a protected font). Available beginning with ATSUI 1.0.
kATSUInvalidAttributeValueErr	-8797	The attribute value is invalid or undefined. Available beginning with ATSIU 1.0.
kATSUInvalidAttributeSizeErr	-8798	The allocated attribute value size is less than required. Available beginning with ATSUI 1.0.
kATSUInvalidAttributeTagErr	-8799	The tag is an ATSUI-reserved value or the wrong type of attribute tag (that is, a style run attribute tag instead of text layout attribute tag and vice versa). Available beginning with ATSUI 1.0.
kATSUInvalidCacheErr	-8800	Indicates an attempt to read style data from an invalid cache (that is, the format of the cached data does not match that used by ATSUI or the cached data is corrupt). Available beginning with ATSUI 1.0.

Result Code	Value	Description
kATSUNotSetErr	-8801	The ATSUI style object's attribute, font feature, font variation is not set; the ATSUI text layout object or single line's attribute is not set; or a font name is not set. Available beginning with ATSUI 1.0.
kATSUNoStyleRunsAssignedErr	-8802	No style runs are assigned to the ATSUI text layout object. Available beginning with ATSUI 1.1.
kATSUQuickDrawTextErr	-8803	The QuickDraw function DrawText encountered an error rendering or measuring a line of text. Available beginning with ATSUI 1.1.
kATSULowLevelErr	-8804	Apple Type Services (ATS) encountered an error while performing an operation requested by ATSUI. Available beginning with ATSUI 1.1.
kATSUNoFontCmapAvailableErr	-8805	The 'CMAP' table cannot be accessed or synthesized for a font. Available beginning with ATSUI 1.1.
kATSUNoFontScalerAvailableErr	-8806	There is no font scaler available for a font. Available beginning with ATSUI 1.1.
kATSUCoordinateOverflowErr	-8807	The coordinate values passed to the function caused a coordinate overflow (greater than 32 KB). Available beginning with ATSUI 1.1.
kATSULineBreakInWord	-8808	The function ATSUBreakLine performed a line break within a word. Available beginning with ATSUI 1.2.
kATSUBusyObjectErr	-8809	An ATSUI object is being used by another thread. Available in Mac OS X v10.1 and later.
kATSUInvalidFontFallbacksErr	-8900	The ATSUFontFallback object is not initialized or is otherwise in an in valid state. Available in Mac OS X v10.1 and later.
kATSUUnsupportedStreamFormatErr	-8901	The data-flattening format is invalid or is not supported by this version of ATSUI. Available in Mac OS X v10.2 and later.
kATSUBadStreamErr	-8902	The data is not formatted as specified by the data-flattening format constant, or the data is corrupt. Available in Mac OS X v10.2 and later.

Result Code	Value	Description
kATSUOutputBufferTooSmallErr	-8903	The output buffer is too small to contain the data output by the function. Available in Mac OS X v10.2 and later.
kATSUInvalidCallInsideCallbackErr	-8904	Your callback is making a call that could cause an infinite recursion. Available in Mac OS X v10.2 and later.
kATSULastErr	-8959	No ATSUI-related result codes may exceed this value. Result code values between kATSUInvalidTextLayoutErr and kATSULastErr are reserved. Available beginning with ATSUI 1.0.

Gestalt Constants

You can check for version and feature availability information by using the ATSUI attribute and version selectors defined in the Gestalt Manager. For more information see Gestalt Manager Reference.

Carbon Accessibility Reference

Framework: ApplicationServices/HIServices.h, Carbon/HIToolbox.h

Overview

This document describes the Carbon accessibility API. You use this API to make your Carbon application accessible to assistive applications and technologies, a process called access enabling.

Who Should Read This Document?

All Carbon application developers should read this document for information on specific functions and constants they may need to access-enable their applications. If you're unsure which parts of the Carbon accessibility API you need, or if you're new to accessibility in Mac OS X, be sure to read the documents listed in "See Also" (page 2073).

Organization of This Document

This document contains API reference in the following sections:

- "Accessibility Object Functions" (page 2074) documents the functions some Carbon applications use to create and manipulate accessibility objects.
- "Accessibility Constants" (page 2081) documents accessibility Carbon events and the constants that define the accessibility event parameters, object attributes, and notifications.
- "Carbon Accessibility Result Codes" (page 2121) describes some of the error codes returned by the Carbon accessibility implementation.

See Also

For more information on accessibility in general and access enabling Carbon applications in particular, you should read the following documents:

- Getting Started With Accessibility
- Accessibility Overview
- Accessibility Programming Guidelines for Carbon

Functions

This section describes the functions some Carbon developers may need to use to access-enable their applications.

AXNotificationHIObjectNotify

Posts a notification for an accessibility object.

```
void AXNotificationHIObjectNotify (
   CFStringRef inNotification,
   HIObjectRef inHIObject,
   UInt64 inIdentifier
);
```

Parameters

inNotification

The string containing the name of the notification to broadcast.

inHIObject

The `HIObjectRef` portion of the accessibility object for which this notification applies.

inIdentifier

The 64-bit identifier portion of the accessibility object for which this notification applies.

Discussion

You use the `AXNotificationHIObjectNotify` function to broadcast changes in an accessibility object to assistive applications. For example, an accessibility object may want to broadcast that the window it represents has moved, or that an attribute value has changed. See "Notifications" (page 2117) for a list of possible notification constants. Note that accessibility objects representing standard user interface elements automatically send out notifications. In general, you do not need to post your own notifications unless you implement custom user interface elements.

Availability

Available in Mac OS X version 10.2 and later.

Not available to 64-bit applications.

Declared In

`HIAccessibility.h`

AXUIElementCreateWithHIObjectAndIdentifier

Creates an accessibility object that represents a user interface element.

```
AXUIElementRef AXUIElementCreateWithHIObjectAndIdentifier (
   HIObjectRef inHIObject,
   UInt64 inIdentifier
);
```

Parameters

inHIObject

A reference to the user interface element this accessibility object represents. You must pass one of the following reference types: `WindowRef`, `ControlRef`, `MenuRef`, or an `HIObjectRef`.

inIdentifier
A 64-bit identifier to uniquely identify the accessibility object within the user interface element. Pass `0` to indicate the base object identified by the `inHIObject` parameter.

Return Value
The newly created accessibility object.

Discussion
If the accessibility object represents part of the substructure of a user interface element, then you must assign it a unique, nonzero identifier value. If the accessibility object represents a complex user interface object as a whole, you must give it the identifier value `0`. For example, a segmented view containing five buttons can have six accessibility objects associated with it:

- The segmented view as a whole, identified by its control reference (`ControlRef`) and identifier value `0`.
- The five button elements, identified by the segmented view reference and identifiers `1` through `5`, respectively.

The accessibility object is a `CFTypeRef` object. You can use `CFEqual` to compare two accessibility objects. You must call `CFRelease` on the accessibility object when you no longer need it.

Availability
Available in Mac OS X version 10.2 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

AXUIElementGetHIObject

Gets the user interface element the given accessibility object represents.

```
HIObjectRef AXUIElementGetHIObject (
   AXUIElementRef inUIElement
);
```

Parameters
inHIAccObj
The accessibility object whose user interface element you want to get.

Return Value
A reference to the user interface element associated with the passed-in accessibility object (or `NULL` if `inHIAccObj` is not a valid accessibility object).

Availability
Available in Mac OS X version 10.2 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

AXUIElementGetIdentifier

Gets the unique identifier associated with an accessibility object.

```
void AXUIElementGetIdentifier (
   AXUIElementRef inUIElement,
   UInt64 *outIdentifier
);
```

Parameters

inHIAccObj

The accessibility object whose identifier you want to get.

outIdentifier

A pointer to a 64-bit integer. On return, `outIdentifier` contains the accessibility object's identifier. If `inHIAccObj` is not a valid accessibility object, this function returns 0. Note that 0 is a valid identifier value, so you should not assume that `inHIAccObj` is invalid if you receive a 0 result.

Discussion

If you create your own accessibility objects to represent custom user interface elements or subviews, you can use the identifier this function returns to identify which accessibility object is being referenced.

Availability

Available in Mac OS X version 10.2 and later.

Not available to 64-bit applications.

Declared In

`HIAccessibility.h`

HICopyAccessibilityActionDescription

Returns the system-defined action description string for a standard action.

```
CFStringRef HICopyAccessibilityActionDescription (
   CFStringRef inAction
);
```

Parameters

inAction

The action for which you want the system-defined description. See "Actions" (page 2115) for the action strings you can use.

Return Value

The system-defined description for the action. When you are finished with the CFString containing the description, you must use `CFRelease` to release it. If you pass in an unsupported action, the results are undefined.

Discussion

The `HICopyAccessibilityActionDescription` function is a convenience function you can use to get the current, system-defined action description for a given action. If you create an accessibility object that supports an action, you must supply the action description. Using this function allows you to take advantage of any changes or enhancements Apple might make.

Availability

Available in Mac OS X version 10.4 and later.

Not available to 64-bit applications.

Declared In
HIAccessibility.h

HICopyAccessibilityRoleDescription

Returns the system-defined role description string for a standard role or role-subrole pair.

```
CFStringRef HICopyAccessibilityRoleDescription (
   CFStringRef inRole,
   CFStringRef inSubrole
);
```

Parameters

inRole

The role for which you want the system-defined description. See "Roles" (page 2087) for the role strings you can use.

inSubrole

The subrole for which you want the system-defined description. See "Subroles" (page 2093) for the subrole strings you can use. Pass `NULL` if your accessible object does not have a subrole.

Return Value
The system-defined description for the role or role-subrole pair. When you are finished with the CFString containing the description, you must use `CFRelease` to release it. If there is no system-defined role description associated with the role or role-subrole pair you pass in, this function returns `NULL`. If you pass in an unknown role or an unknown subrole, this function returns `NULL`.

Discussion
The `HICopyAccessibilityRoleDescription` function is a convenience function you can use if you have to provide the role description for an accessibility object you create. Instead of hard-coding a role description for an accessibility object, you should use this function to get the current, system-defined role description. This allows you to take advantage of any changes or enhancements Apple might make.

Availability
Available in Mac OS X version 10.4 and later.
Not available to 64-bit applications.

Declared In
HIAccessibility.h

HIObjectIsAccessibilityIgnored

Returns whether the given HIObject is marked as ignored for accessibility purposes.

```
Boolean HIObjectIsAccessibilityIgnored (
   HIObjectRef inObject
);
```

Parameters

inObject

The object whose accessibility ignored state you wish to query.

Return Value
A Boolean value indicating whether the HIObject is ignored for accessibility purposes.

Availability
Available in Mac OS X version 10.2 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

HIObjectOverrideAccessibilityContainment

Allows you to override the accessibility objects an HIObject would usually supply as the values of its `kAXParentAttribute`, `kAXWindowAttribute`, and `kAXTopLevelUIElementAttribute` attributes.

```
OSStatus HIObjectOverrideAccessibilityContainment (
   HIObjectRef inHIObject,
   AXUIElementRef inDesiredParent,
   AXUIElementRef inDesiredWindow,
   AXUIElementRef inDesiredTopLevelUIElement
);
```

Parameters

inHIObject
The `HIObjectRef` whose parent attribute you want to override.

inDesiredParent
The `AXUIElementRef` that you want the given HIObject to return as the value of its `kAXParentAttribute` attribute. This function makes a copy of the `AXUIElementRef` and you must release the *inDesiredParent* parameter after you call this function. Passing `NULL` in this parameter indicates you do not want the HIObject to override the value of its `kAXParentAttribute` attribute.

inDesiredWindow
The `AXUIElementRef` that you want the given HIObject to return as the value of its `kAXWindowAttribute` attribute. This function makes a copy of the `AXUIElementRef` and you must release the *inDesiredWindow* parameter after you call this function. Passing `NULL` in this parameter indicates you do not want the HIObject to override the value of its `kAXWindowAttribute` attribute (if the value exists).

inDesiredTopLevelUIElement
The `AXUIElementRef` that you want the given HIObject to return as the value of its `kAXTopLevelUIElementAttribute` attribute. This function makes a copy of the `AXUIElementRef` and you must release the *inDesiredTopLevelUIElement* parameter after you call this function. Passing `NULL` in this parameter indicates you do not want the HIObject to override the value of its `kAXTopLevelUIElementAttribute` attribute (if the value exists).

Return Value
An OSStatus result code. If the `HIObjectRef` is invalid, this function returns `paramErr`.

Discussion
This function allows you to change the parent that the given HIObject would usually supply to the accessibility hierarchy. For example, you might call this function on the menu of a pop-up control to ensure that the menu returns the pop-up control as its parent (rather than the application). Optionally, this function also allows you to change the window and top-level accessibility object the given HIObject would supply.

If the input HIObject is a standard toolbox object, such as an HIView or a menu, the input HIObject will not be included as an accessibility child of its normal parent. In all other cases, it is the client's responsibility to ensure that the input HIObject is not included as an accessibility child of its normal parent.

If the desired `AXUIElementRef` parent represents an HIView, a menu, or a window, the input HIObject will be included automatically as an accessibility child of the specified parent. In all other cases, it is the client's responsibility to manually include the input HIObject as an accessibility child of the specified parent. To represent an HIView, a menu, or a window, an `AXUIElementRef` must contain the appropriate `HIObjectRef`, as well as an identifier value of 0.

Note that similar rules don't apply to the handling of the window and top-level element attributes, because those attributes don't represent two-way relationships.

Not every type of HIObject supports a containment override; currently, HIViews, menus, and windows support containment overrides.

Availability
Available in Mac OS X version 10.4 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

HIObjectSetAccessibilityIgnored

Marks an HIObject as ignored or unignored for accessibility purposes.

```
OSStatus HIObjectSetAccessibilityIgnored (
   HIObjectRef inObject,
   Boolean inIgnored
);
```

Parameters

inObject
The object whose accessibility ignored state you wish to change.

inIgnored
A Boolean value indicating whether to ignore the object (`TRUE`) or not (`FALSE`).

Return Value
An OSStatus signifying success or failure.

Discussion
An ignored HIObject is not shown to an assistive application that uses the accessibility APIs to examine the interface of your application. Your application's accessibility implementation should still report an ignored HIObject as usual. The Carbon accessibility implementation automatically hides ignored HIObjects from assistive applications.

Note: By default, an HIObject is *not* ignored.

Availability
Available in Mac OS X version 10.2 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

HIObjectSetAuxiliaryAccessibilityAttribute

Associates an additional accessibility attribute with an accessibility object (a UIElement) that is used to represent a given HIObject or part thereof.

```
OSStatus HIObjectSetAuxiliaryAccessibilityAttribute (
   HIObjectRef inHIObject,
   UInt64 inIdentifier,
   CFStringRef inAttributeName,
   CFTypeRef inAttributeData
);
```

Parameters

inHIObject

The `HIObjectRef` portion of the object-identifier pair to which the attribute data is associated.

inIdentifier

The 64-bit identifier portion of the object-identifier pair to which the attribute data is associated. Pass 0 in this parameter when you want to associate the attribute data to the HIObject as a whole. You might do this if, for example, you want to give a description attribute to the object representing a button.

inAttributeName

A `CFStringRef` of the name of the attribute. This string is retained before it is added to the auxiliary attribute storage area.

inAttributeData

A `CFTypeRef` containing the data supplied for the attribute's value. This data is retained before it is added to the auxiliary attribute storage area; you may release this data after calling this function. If you pass `NULL` in this parameter, it indicates that the named auxiliary attribute should no longer be associated with the object-identifier pair and any named attribute data previously associated with the object-identifier pair will be released.

Return Value

An OSStatus result code. The function returns `noErr` if it was able to associate the attribute data with the HIObject. If the `HIObjectRef` is invalid, `paramErr` is returned.

Discussion

This function allows your application to provide the name of and data for an accessibility attribute you want to add to the UIElement that represents a given HIObject-identifier pair. Normally, accessibility attributes are only supplied dynamically through Carbon events, but this function allows you to supply them statically.

This function only allows you to associate attributes whose values never change. If you need to supply attribute whose values are determined dynamically or whose values are settable, you must install the necessary Carbon accessibility event handlers. See Accessibility Programming Guidelines for Carbon for more information about how this works.

This function is particularly useful for supplying the values of the `kAXDescriptionAttribute`, `kAXTitleUIElementAttribute`, `kAXServesAsTitleForUIElementsAttribute`, `kAXLinkedUIElementsAttribute` attributes and other attributes whose values are specific to the layout and usage of your application.

The auxiliary attribute store (containing attribute values you supply using this function) is consulted during the HIObject's default handling of the Carbon accessibility attribute events. Therefore, any programmatic handling of a given accessibility attribute has the opportunity to override or block the consultation of the store. In general, if the toolbox or a Carbon event handler can provide the attribute value some other way, the store is not consulted.

Availability
Available in Mac OS X version 10.4 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

Constants

This section describes the constants that define accessibility events and aspects of accessibility objects. The accessibility event constants are defined in `CarbonEvents.h` in the Carbon framework. The accessibility object constants are defined in header files in the ApplicationServices framework.

Accessibility Events

Accessibility Event Constants

Define accessibility events (`kEventClassAccessibility`).

```
enum
{
    kEventAccessibleGetChildAtPoint = 1,
    kEventAccessibleGetFocusedChild = 2,
    kEventAccessibleGetAllAttributeNames = 21,
    kEventAccessibleGetAllParameterizedAttributeNames = 25,
    kEventAccessibleGetNamedAttribute = 22,
    kEventAccessibleSetNamedAttribute = 23,
    kEventAccessibleIsNamedAttributeSettable = 24,
    kEventAccessibleGetAllActionNames = 41,
    kEventAccessiblePerformNamedAction = 42,
    kEventAccessibleGetNamedActionDescription = 44
};
```

Constants
`kEventAccessibleGetChildAtPoint`

A request sent by an assistive application to get the accessible child of the given accessibility object that contains the given point. The `kEventParamMouseLocation` parameter contains the location in global coordinates.

If you handle this event, you use the `kEventParamAccessibleChild` parameter to return an accessible first-order child of the accessibility object receiving the event. If there is no child at the given point, your handler should leave the `kEventParamAccessibleChild` parameter empty and return `noErr`. You must not set the `kEventParamAccessibleChild` parameter to a grandchild or more distant descendant of the accessible object receiving this event.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessibleGetFocusedChild`

A request sent by an assistive application to get the accessible child of the given accessibility object that is part of the focus chain.

If you handle this event, you set the `kEventParamAccessibleChild` parameter to a first-order, accessible child that is focused or is the ancestor of a focused object. If there is no child in the focus chain, your handler should leave the `kEventParamAccessibleChild` parameter empty and return `noErr`. You must not set the `kEventParamAccessibleChild` parameter to a grandchild or more distant descendant of the accessible object receiving this event.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessibleGetAllAttributeNames`

A request sent by an assistive application to get the names of all attributes the given accessibility object supports.

If you handle this event, you create a CFString object for the name of each non-parameterized attribute and add it to the mutable array in the `kEventParamAccessibleAttributeNames` parameter. If the accessibility object receiving the event supports parameterized attributes, you return them in the handler for the `kEventAccessibleGetAllParameterizedAttributeNames` event.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessibleGetAllParameterizedAttributeNames`

A request sent by an assistive application to get the names of all parameterized attributes the given accessibility object supports.

If you handle this event, you create a CFString object for the name of each parameterized attribute and add it to the mutable array in the `kEventParamAccessibleAttributeNames` parameter. You must not return any regular, non-parameterized attribute names with this event. Instead, return regular attribute names in the handler for the `kEventAccessibleGetAllAttributeNames` event.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.3 and later.

`kEventAccessibleGetNamedAttribute`

A request sent by an assistive application to get the value of the given attribute.

If you handle this event, you determine if the given accessibility object supports the attribute named in the `kEventParamAccessibleAttributeName` parameter. If it does, you return the attribute's value in the `kEventParamAccessibleAttributeValue` parameter. If the accessibility object does not support the attribute, return the `eventNotHandledErr` error. The type of the `kEventParamAccessibleAttributeValue` parameter varies with the type of the attribute's value.

This event may also include the optional `kEventParamAccessibleAttributeParameter` parameter that describes the parameters of a parameterized attribute. Note that parameterized attributes were introduced in Mac OS X version 10.3.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessibleSetNamedAttribute`

A request sent by an assistive application to set the value of the given attribute to the passed-in value.

If you handle this event, you determine if the given accessibility object supports the attribute named in the `kEventParamAccessibleAttributeName` parameter and if the attribute is settable. Then, you set the named attribute's value to the value supplied in the `kEventParamAccessibleAttributeValue` parameter. If you cannot handle this event (because, for example, the accessibility object does not support this attribute, the attribute is not settable, or the value is not appropriate), return the `eventNotHandledErr` error.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessibleIsNamedAttributeSettable`

A request sent by an assistive application to find out if the given attribute's value can be changed.

If you handle this event, you determine if the accessibility object supports the given attribute. If it does, you return a Boolean value in the `kEventParamAccessibleAttributeSettable` parameter that indicates whether the attribute's value can be changed.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessibleGetAllActionNames`

Sent by an assistive application to find out which actions the given accessibility object supports.

If you handle this event, you create a CFString object for the name of each action the given accessibility object supports and add it to the mutable array in the `kEventParamAccessibleActionNames` parameter.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessiblePerformNamedAction`

Sent by an assistive application when it wants the given accessibility object to perform the given action.

If you handle this event, you determine if the accessibility object supports the action named in the `kEventParamAccessibleActionName` parameter. If it does, you perform it.

In Mac OS X version 10.3 and later, this event includes the `kEventParamAccessibilityEventQueued` parameter, which indicates whether the event was queued. You check the value of this parameter before you perform an action that might result in a call to a routine that may not return immediately. If the event is queued, you can perform such an action without the possibility of causing an assistive application to receive a time-out error waiting for the action to complete. If the event is not queued, your handler can return the `eventDeferAccessibilityEventErr` to request that it be queued and sent to you later.

In versions of Mac OS X prior to 10.3, events are always directly dispatched and you should perform a requested action even if it might cause an assistive application to receive a time-out error.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

`kEventAccessibleGetNamedActionDescription`

Sent by an assistive application to get the human-intelligible name of the given action.

If you handle this event, you determine if the given accessibility object supports the given action. If it does, you return the value of the action's description property in the `kEventParamAccessibleActionDescription` parameter. To do this, you do not create a CFString object for the action description. Instead, you must modify the mutable string object in the `kEventParamAccessibleActionDescription` parameter to contain the action description.

Declared in `HIAccessibility.h`.

Available in Mac OS X version 10.2 and later.

Discussion

Table 47-1 (page 2084) shows the parameters related to accessibility events.

Table 47-1 Parameter names and types for accessibility event kinds

Event kind	Parameter name	Parameter type
`kEventAccessible-GetChildAtPoint`	`kEventParamAccessibleObject`	`typeCFTypeRef` (an `AXUIElementRef`)
	`kEventParamAccessibleMouse-Location`	`typeHIPoint`
	`kEventParamAccessibleChild`	`typeCFTypeRef`
`kEventAccessible-GetFocusedChild`	`kEventParamAccessibleObject`	`typeCFTypeRef` (an `AXUIElementRef`)
	`kEventParamAccessibleChild`	`typeCFTypeRef`
`kEventAccessibleGet-AllAttributeNames`	`kEventParamAccessibleObject`	`typeCFTypeRef` (an `AXUIElementRef`)
	`kEventParamAccessibleAttribute-Names`	`typeCFMutableArrayRef`
`kEventAccessibleGet-AllParameterized-AttributeNames`	`kEventParamAccessibleObject`	`typeCFTypeRef` (an `AXUIElementRef`)
	`kEventParamAccessibleAttribute-Names`	`typeCFMutableArrayRef`
`kEventAccessibleGet-NamedAttribute`	`kEventParamAccessibleObject`	`typeCFTypeRef` (an `AXUIElementRef`)
	`kEventParamAccessibleAttribute-Name`	`typeCFStringRef`
	`kEventParamAccessibleAttribute-Parameter` (Optional; introduced in Mac OS X version 10.3)	`typeCFTypeRef`

Event kind	Parameter name	Parameter type
	kEventParamAccessibleAttribute-Value	typeCFTypeRef (Varies with the type of the attribute value)
kEventAccessibleSet-NamedAttribute	kEventParamAccessibleObject	typeCFTypeRef (an AXUIElementRef)
	kEventParamAccessibleAttribute-Name	typeCFStringRef
	kEventParamAccessibleAttribuute-Value	typeCFTypeRef (Varies with the type of the attribute value)
kEventAccessible-IsNamedAttribute-Settable	kEventParamAccessibleObject	typeCFTypeRef (an AXUIElementRef)
	kEventParamAccessibleAttribute-Name	typeCFStringRef
	kEventParamAccessibleAttribute-Settable	typeBoolean
kEventAccessibleGet-AllActionNames	kEventParamAccessibleObject	typeCFTypeRef
	kEventParamAccessibleActionNames	typeCFMutableArrayRef
kEventAccessible-PerformNamedAction	kEventParamAccessibleObject	typeCFTypeRef (an AXUIElementRef)
	kEventParamAccessibleActionName	typeCFStringRef
	kEventParamAccessibleEventQueued (Only in Mac OS X version 10.3 and later)	typeBoolean
kEventAccessible-GetNamedAction-Description	kEventParamAccessibleObject	typeCFTypeRef (an AXUIElementRef)
	kEventParamAccessibleActionName	typeCFStringRef
	kEventParamAccessibleAction-Description	typeCFMutableStringRef

Accessibility Event Parameters

Define parameters related to accessibility events.

```
enum
{
    kEventParamAccessibleObject = 'aojb',
    kEventParamAccessibleChild = 'achl',
    kEventParamAccessibleAttributeName = 'atnm',
    kEventParamAccessibleAttributeNames = 'atns',
    kEventParamAccessibleAttributeValue = 'atvl',
    kEventParamAccessibleAttributeSettable = 'atst',
    kEventParamAccessibleAttributeParameter = 'atpa',
    kEventParamAccessibleActionName = 'acnm',
    kEventParamAccessibleActionNames = 'acns',
    kEventParamAccessibleActionDescription = 'acds',
    kEventParamAccessibleEventQueued = 'aequ'
};
```

Constants

`kEventParamAccessibleObject`
Specifies an accessibility object. The parameter type is `typeCFTypeRef`, and the data must be of type `AXUIElementRef`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleChild`
Specifies the child accessibility object. The parameter type is `typeCFTypeRef`, and the data must be of type `AXUIElementRef`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleAttributeName`
Specifies an attribute name. The parameter type is `typeCFStringRef`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleAttributeNames`
Specifies an array of attribute names (each of type `CFStringRef`). The parameter type is `typeCFMutableArrayRef`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleAttributeValue`
Specifies the value of an attribute. The parameter type varies according to the attribute. However, this value must be one of the flat data types, such as point, rectangle, integer, float, or any CFType, and must be able to be packaged in a CFPropertyList. In particular, the data should not be a pointer, because you cannot be sure how long an assistive application will retain the value, or in what way it will interpret the value.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleAttributeSettable`
Specifies whether an attribute is settable. The parameter type is `typeBoolean`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleAttributeParameter`
Specifies the parameters of a parameterized attribute. The parameter type is `typeCFTypeRef`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleActionName`
Specifies an action name. The parameter type is `typeCFStringRef`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleActionNames`
Specifies an array of action names (each of type `CFStringRef`) . The parameter type is `typeCFMutableArrayRef`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleActionDescription`
Specifes the description of an action. The parameter type is `typeCFMutableStringRef`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventParamAccessibleEventQueued`
Specifies whether the event has been queued. The parameter type is `typeBoolean`.

Accessibility Event Class

Defines the event class for accessibility events.

```
enum
{
    kEventClassAccessibility = 'acce',
};
```

Constants

`kEventClassAccessibility`
Pass this value for the event class when registering for accessibility events.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

Accessibility Object Constants

Roles

Define the values an accessibility object's role attribute can have.

```
#define kAXApplicationRole  CFSTR("AXApplication")
#define kAXSystemWideRole   CFSTR("AXSystemWide")
#define kAXWindowRole       CFSTR("AXWindow")
#define kAXSheetRole        CFSTR("AXSheet")
#define kAXDrawerRole       CFSTR("AXDrawer")
#define kAXGrowAreaRole     CFSTR("AXGrowArea")
#define kAXImageRole        CFSTR("AXImage")
#define kAXUnknownRole      CFSTR("AXUnknown")
#define kAXButtonRole       CFSTR("AXButton")
#define kAXRadioButtonRole  CFSTR("AXRadioButton")
#define kAXCheckBoxRole     CFSTR("AXCheckBox")
#define kAXPopUpButtonRole  CFSTR("AXPopUpButton")
#define kAXMenuButtonRole   CFSTR("AXMenuButton")
#define kAXTabGroupRole     CFSTR("AXTabGroup")
#define kAXTableRole        CFSTR("AXTable")
#define kAXColumnRole       CFSTR("AXColumn")
#define kAXRowRole          CFSTR("AXRow")
#define kAXOutlineRole      CFSTR("AXOutline")
#define kAXBrowserRole      CFSTR("AXBrowser")
#define kAXScrollAreaRole   CFSTR("AXScrollArea")
#define kAXScrollBarRole    CFSTR("AXScrollBar")
#define kAXRadioGroupRole   CFSTR("AXRadioGroup")
#define kAXListRole         CFSTR("AXList")
#define kAXGroupRole        CFSTR("AXGroup")
#define kAXValueIndicatorRole CFSTR ("AXValueIndicator")
#define kAXComboBoxRole     CFSTR("AXComboBox")
#define kAXSliderRole       CFSTR("AXSlider")
#define kAXIncrementorRole  CFSTR("AXIncrementor")
#define kAXBusyIndicatorRole CFSTR ("AXBusyIndicator")
#define kAXProgressIndicatorRole  CFSTR("AXProgressIndicator")
#define kAXRelevanceIndicatorRole CFSTR("AXRelevanceIndicator")
#define kAXToolbarRole      CFSTR("AXToolbar")
#define kAXDisclosureTriangleRole CFSTR("AXDisclosureTriangle")
#define kAXTextFieldRole CFSTR("AXTextField")
#definekAXTextAreaRole      CFSTR("AXTextArea")
#define kAXStaticTextRole   CFSTR("AXStaticText")
#define kAXMenuBarRole      CFSTR("AXMenuBar")
#define kAXMenuBarItemRole  CFSTR("AXMenuBarItem")
#define kAXMenuRole         CFSTR("AXMenu")
#define kAXMenuItemRole     CFSTR("AXMenuItem")
#define kAXSplitGroupRole   CFSTR("AXSplitGroup")
#define kAXSplitterRole     CFSTR("AXSplitter")
#define kAXColorWellRole    CFSTR("AXColorWell")
#define kAXTimeFieldRole    CFSTR("AXTimeField")
#define kAXDateFieldRole    CFSTR("AXDateField")
#define kAXHelpTagRole      CFSTR("AXHelpTag")
#define kAXMatteRole        CFSTR("AXMatteRole")
#define kAXDockItemRole     CFSTR("AXDockItem")
```

Constants

`kAXApplicationRole`

An application.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXSystemWideRole`
The system-wide accessibility object.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXWindowRole`
A window.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXSheetRole`
A sheet.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXDrawerRole`
A drawer.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXGrowAreaRole`
A grow control.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXImageRole`
An image.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXUnknownRole`
Generic role value for an unknown accessibility object.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXButtonRole`
A button.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXRadioButtonRole`
A radio button.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXCheckBoxRole`
A check box.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXPopUpButtonRole`
A pop-up button.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXMenuButtonRole`
A menu button.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXTabGroupRole`
A tab view.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXTableRole`
A table.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXColumnRole`
A column.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXRowRole`
A row.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXOutlineRole`
An accessibility object that displays row-based, hierarchically structured data, such as the list view in a Finder window.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXBrowserRole`
An accessibility object that displays column-based, hierarchically structured data, such as the column view in a Finder window.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXScrollAreaRole`
An accessibility object that displays data managed by scrolling controls.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXScrollBarRole`
A scroll bar control.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXRadioGroupRole`
A set of radio buttons.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXListRole`
A list view.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXGroupRole`
A group box. This role can also be used to group other views without any visual indication of the grouping.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXValueIndicatorRole`
A control that indicates the value of an accessibility object, such as the scroller of a scroll bar control

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXComboBoxRole`
A combo box control.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXSliderRole`
A slider control.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXIncrementorRole`
A stepper control (also known as the “little arrows”).

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXBusyIndicatorRole`
An asynchronous progress indicator.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXProgressIndicatorRole`
A determinate or indeterminate progress indicator.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXRelevanceIndicatorRole`
A relevance indicator.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXToolbarRole`
A toolbar.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXDisclosureTriangleRole`
A disclosure triangle control.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXTextFieldRole`
A text field.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXTextAreaRole`
The editable text area in a control or window.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXStaticTextRole`
A string of static text displayed in a window that is not part of any control.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXMenuBarRole`
A menu bar.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXMenuBarItemRole`
A menu bar item.
Available in Mac OS X v10.3 and later.
Declared in `AXRoleConstants.h`.

`kAXMenuRole`
A menu.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXMenuItemRole`
A menu item.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXSplitGroupRole`
A split view.
Available in Mac OS X v10.2 and later.
Declared in `AXRoleConstants.h`.

`kAXSplitterRole`
A splitter bar control.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXColorWellRole`
A color well.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXTimeFieldRole`
A field that displays time.

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXDateFieldRole`
A field that displays dates.

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXHelpTagRole`
A help tag.

Available in Mac OS X v10.4 and later.

Declared in `AXRoleConstants.h`.

`kAXMatteRole`
The outer view that represents the entire contents, including the view through the matte hole, the contents hidden by the matte frame, and the resizing and repositioning controls. An example of an object with a matte role is the iChat icon scaling window.

Available in Mac OS X v10.4 and later.

Declared in `AXRoleConstants.h`.

`kAXDockItemRole`
An icon that represents an item in the Dock.

Available in Mac OS X v10.4 and later.

Declared in `AXRoleConstants.h`.

Discussion
The value of the role attribute describes what the object is, not what it does. See the "Roles and Associated Attributes" appendix in Accessibility Overview for more information on which attributes are associated with each role.

Subroles

Define the values for an accessibility object's subrole attribute.

```
#define kAXCloseButtonSubrole       CFSTR("AXCloseButton")
#define kAXMinimizeButtonSubrole    CFSTR("AXMinimizeButton")
#define kAXZoomButtonSubrole        CFSTR("AXZoomButton")
#define kAXToolbarButtonSubrole     CFSTR("AXToolbarButton")
#define kAXSecureTextFieldSubrole   CFSTR("AXSecureTextField")
#define kAXTableRowSubrole          CFSTR("AXTableRow")
#define kAXOutlineRowSubrole        CFSTR("AXOutlineRow")
#define kAXUnknownSubrole           CFSTR("AXUnknown")
#define kAXStandardWindowSubrole    CFSTR("AXStandardWindow")
#define kAXDialogSubrole            CFSTR("AXDialog")
#define kAXSystemDialogSubrole      CFSTR("AXSystemDialog")
#define kAXFloatingWindowSubrole    CFSTR("AXFloatingWindow")
#define kAXSystemFloatingWindowSubrole CFSTR("AXSystemFloatingWindow")
#define kAXIncrementArrowSubrole    CFSTR("AXIncrementArrow")
#define kAXDecrementArrowSubrole    CFSTR("AXDecrementArrow")
#define kAXIncrementPageSubrole     CFSTR("AXIncrementPage")
#define kAXDecrementPageSubrole     CFSTR("AXDecrementPage")
#define kAXSortButtonSubrole        CFSTR("AXSortButton")
#define kAXSearchFieldSubrole       CFSTR("AXSearchField")
#define kAXApplicationDockItemSubrole CFSTR("AXApplicationDockItem")
#define kAXDocumentDockItemSubrole  CFSTR("AXDocumentDockItem")
#define kAXFolderDockItemSubrole    CFSTR("AXFolderDockItem")
#define kAXMinimizedWindowDockItemSubrole CFSTR("AXMinimizedWindowDockItem")
#define kAXURLDockItemSubrole       CFSTR("AXURLDockItem")
#define kAXDockExtraDockItemSubrole CFSTR("AXDockExtraDockItem")
#define kAXTrashDockItemSubrole     CFSTR("AXTrashDockItem")
#define kAXProcessSwitcherListSubrole CFSTR("AXProcessSwitcherList")
```

Constants

`kAXCloseButtonSubrole`

A close button (that is, the red button in a window's title bar that closes the window).

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXMinimizeButtonSubrole`

A minimize button (that is, the yellow button in a window's title bar that minimizes the window into the Dock).

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXZoomButtonSubrole`

A zoom button (that is, the green button in a window's title bar that adjusts the window's size).

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXToolbarButtonSubrole`

A toolbar button (that is, the button in a window's title bar that hides and reveals the toolbar).

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXSecureTextFieldSubrole`

A text field intended to contain sensitive data and that displays the user's input as a series of bullets.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXTableRowSubrole`
A row in a table.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXOutlineRowSubrole`
A row in an outline view (see `kAXOutlineRole` for a description of an outline view).

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXUnknownSubrole`
A subrole for an unknown type of window. A window should include a subrole to further define its type. If your window does not conform to an existing subrole, you can use the unknown subrole. Alternatively, you can return the `eventNotHandledErr` error when your window is asked for its subrole.

Available in Mac OS X v10.2 and later.

Declared in `AXRoleConstants.h`.

`kAXStandardWindowSubrole`
A standard window that includes a title bar (that is, not an inspector window or a sheet).

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXDialogSubrole`
A dialog window, such as an alert.

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXSystemDialogSubrole`
A system-generated dialog window that floats on the top layer, regardless of which application is frontmost. Use this subrole only when a dialog or alert applies to the system as a whole, such as a shutdown dialog.

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXFloatingWindowSubrole`
A utility window.

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXSystemFloatingWindowSubrole`
A system-generated utility window.

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXIncrementArrowSubrole`
The up arrow of a scroll bar.

Available in Mac OS X v10.3 and later.

Declared in `AXRoleConstants.h`.

`kAXDecrementArrowSubrole`
The down arrow of a scroll bar.
Available in Mac OS X v10.3 and later.
Declared in `AXRoleConstants.h`.

`kAXIncrementPageSubrole`
The increment area in the scroll track of a scroll bar.
Available in Mac OS X v10.3 and later.
Declared in `AXRoleConstants.h`.

`kAXDecrementPageSubrole`
The decrement area in the scroll track of a scroll bar.
Available in Mac OS X v10.3 and later.
Declared in `AXRoleConstants.h`.

`kAXSortButtonSubrole`
A column heading button in a list or column view.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXSearchFieldSubrole`
A search field.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXApplicationDockItemSubrole`
An icon in the Dock that represents an application.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXDocumentDockItemSubrole`
An icon in the Dock that represents a document.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXFolderDockItemSubrole`
An icon in the Dock that represents a folder.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXMinimizedWindowDockItemSubrole`
An icon in the Dock that represents a minimized window.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXURLDockItemSubrole`
An icon in the Dock that represents a URL.
Available in Mac OS X v10.4 and later.
Declared in `AXRoleConstants.h`.

`kAXDockExtraDockItemSubrole`
: An icon in the Dock that represents a Dock Extra.

  Available in Mac OS X v10.4 and later.

  Declared in `AXRoleConstants.h`.

`kAXTrashDockItemSubrole`
: The icon in the Dock that represents the Trash.

  Available in Mac OS X v10.4 and later.

  Declared in `AXRoleConstants.h`.

`kAXProcessSwitcherListSubrole`
: The display of running applications (processes) that appears when a user presses Command-Tab.

  Available in Mac OS X v10.4 and later.

  Declared in `AXRoleConstants.h`.

Discussion
A subrole provides a more specific description of an accessibility object's role. If an accessibility object is of a well-defined subtype, it can include the subrole attribute to provide additional information to an assistive application.

Attributes

Define the attributes available for accessibility objects.

```
//General attributes
#define kAXRoleAttribute                    CFSTR("AXRole")
#define kAXSubroleAttribute                 CFSTR("AXSubrole")
#define kAXRoleDescriptionAttribute         CFSTR("AXRoleDescription")
#define kAXHelpAttribute                    CFSTR("AXHelp")
#define kAXTitleAttribute                   CFSTR("AXTitle")
#define kAXValueAttribute                   CFSTR("AXValue")
#define kAXMinValueAttribute                CFSTR("AXMinValue")
#define kAXMaxValueAttribute                CFSTR("AXMaxValue")
#define kAXValueIncrementAttribute          CFSTR("AXValueIncrement")
#define kAXAllowedValuesAttribute           CFSTR("AXAllowedValues")
#define kAXEnabledAttribute                 CFSTR("AXEnabled")
#define kAXFocusedAttribute                 CFSTR("AXFocused")
#define kAXParentAttribute                  CFSTR("AXParent")
#define kAXChildrenAttribute                CFSTR("AXChildren")
#define kAXSelectedChildrenAttribute        CFSTR("AXSelectedChildren")
#define kAXVisibleChildrenAttribute         CFSTR("AXVisibleChildren")
#define kAXWindowAttribute                  CFSTR("AXWindow")
#define kAXTopLevelUIElementAttribute       CFSTR("AXTopLevelUIElement")
#define kAXPositionAttribute                CFSTR("AXPosition")
#define kAXSizeAttribute                    CFSTR("AXSize")
#define kAXOrientationAttribute             CFSTR("AXOrientation")
#define kAXDescriptionAttribute             CFSTR("AXDescription")
```

```
// Text-specific attributes
#define kAXSelectedTextAttribute            CFSTR("AXSelectedText")
#define kAXVisibleCharacterRangeAttribute CFSTR("AXVisibleCharacterRange")
#define kAXSelectedTextRangeAttribute       CFSTR("AXSelectedTextRange")
#define kAXNumberOfCharactersAttribute      CFSTR("AXNumberOfCharacters")
#define kAXSharedTextUIElementsAttribute    CFSTR("AXSharedTextUIElements")
#define kAXSharedCharacterRangeAttribute    CFSTR("AXSharedCharacterRange")

// Window-specific attributes
#define kAXMainAttribute                    CFSTR("AXMain")
#define kAXMinimizedAttribute               CFSTR("AXMinimized")
#define kAXCloseButtonAttribute             CFSTR("AXCloseButton")
#define kAXZoomButtonAttribute              CFSTR("AXZoomButton")
#define kAXMinimizeButtonAttribute          CFSTR("AXMinimizeButton")
#define kAXToolbarButtonAttribute           CFSTR("AXToolbarButton")
#define kAXGrowAreaAttribute                CFSTR("AXGrowArea")
#define kAXProxyAttribute                   CFSTR("AXProxy")
#define kAXModalAttribute                   CFSTR("AXModal")
#define kAXDefaultButtonAttribute           CFSTR("AXDefaultButton")
#define kAXCancelButtonAttribute            CFSTR("AXCancelButton")

// Menu-specific attributes
#define kAXMenuItemCmdCharAttribute         CFSTR("AXMenuItemCmdChar")
#define kAXMenuItemCmdVirtualKeyAttribute
                                    CFSTR("AXMenuItemCmdVirtualKey")
#define kAXMenuItemCmdGlyphAttribute        CFSTR("AXMenuItemCmdGlyph")
#define kAXMenuItemCmdModifiersAttribute
                                       CFSTR("AXMenuItemCmdModifiers")
#define kAXMenuItemMarkCharAttribute    CFSTR("AXMenuItemMarkChar")
#define kAXMenuItemPrimaryUIElementAttribute CFSTR("AXMenuItemPrimaryUIElement")

// Application-specific attributes
#define kAXMenuBarAttribute                     CFSTR("AXMenuBar")
#define kAXWindowsAttribute                     CFSTR("AXWindows")
#define kAXFrontmostAttribute                   CFSTR("AXFrontmost")
#define kAXHiddenAttribute                      CFSTR("AXHidden")
#define kAXMainWindowAttribute                  CFSTR("AXMainWindow")
#define kAXFocusedWindowAttribute               CFSTR("AXFocusedWindow")
#define kAXFocusedUIElementAttribute        CFSTR("AXFocusedUIElement")
```

```
// Miscellaneous attributes
#define kAXHeaderAttribute                  CFSTR("AXHeader")
#define kAXEditedAttribute                  CFSTR("AXEdited")
#define kAXValueWrapsAttribute              CFSTR("AXValueWraps")
#define kAXTabsAttribute                    CFSTR("AXTabs")
#define kAXTitleUIElementAttribute          CFSTR("AXTitleUIElement")
#define kAXHorizontalScrollBarAttribute     CFSTR("AXHorizontalScrollBar")
#define kAXVerticalScrollBarAttribute       CFSTR("AXVerticalScrollBar")
#define kAXOverflowButtonAttribute          CFSTR("AXOverflowButton")
#define kAXFilenameAttribute                CFSTR("AXFilename")
#define kAXExpandedAttribute                CFSTR("AXExpanded")
#define kAXSelectedAttribute                CFSTR("AXSelected")
#define kAXSplittersAttribute               CFSTR("AXSplitters")
#define kAXNextContentsAttribute            CFSTR("AXNextContents")
#define kAXDocumentAttribute                CFSTR("AXDocument")
#define kAXDecrementButtonAttribute         CFSTR("AXDecrementButton")
#define kAXIncrementButtonAttribute         CFSTR("AXIncrementButton")
#define kAXPreviousContentsAttribute        CFSTR("AXPreviousContents")
#define kAXContentsAttribute                CFSTR("AXContents")
#define kAXIncrementorAttribute             CFSTR("AXIncrementor")
#define kAXHourFieldAttribute               CFSTR("AXHourField")
#define kAXMinuteFieldAttribute             CFSTR("AXMinuteField")
#define kAXSecondFieldAttribute             CFSTR("AXSecondField")
#define kAXAMPMFieldAttribute               CFSTR("AXAMPMField")
#define kAXDayFieldAttribute                CFSTR("AXDayField")
#define kAXMonthFieldAttribute              CFSTR("AXMonthField")
#define kAXYearFieldAttribute               CFSTR("AXYearField")
#define kAXColumnTitleAttribute             CFSTR("AXColumnTitles")
#define kAXURLAttribute                     CFSTR("AXURL")
#define kAXLabelUIElementsAttribute         CFSTR("AXLabelUIElements")
#define kAXLabelValueAttribute              CFSTR("AXLabelValue")
#define kAXShownMenuUIElementAttribute      CFSTR("AXShownMenuUIElement")
#define kAXServesAsTitleForUIElementsAttribute
CFSTR("AXServesAsTitleForUIElements")
#define kAXLinkedUIElementsAttribute CFSTR("AXLinkedUIElements")

// Table and outline view attributes
#define kAXRowsAttribute                    CFSTR("AXRows")
#define kAXVisibleRowsAttribute             CFSTR("AXVisibleRows")
#define kAXSelectedRowsAttribute            CFSTR("AXSelectedRows")
#define kAXColumnsAttribute                 CFSTR("AXColumns")
#define kAXVisibleColumnsAttribute          CFSTR("AXVisibleColumns")
#define kAXSelectedColumnsAttribute         CFSTR("AXSelectedColumns")
#define kAXSortDirectionAttribute           CFSTR("AXSortDirection")
#define kAXColumnHeaderUIElementsAttribute CFSTR("AXColumnHeaderUIElements")
#define kAXIndexAttribute                   CFSTR("AXIndex")
#define kAXDisclosingAttribute              CFSTR("AXDisclosing")
#define kAXDisclosedRowsAttribute           CFSTR("AXDisclosedRows")
#define kAXDisclosedByRowAttribute          CFSTR("AXDisclosedByRow")

// Matte attributes
#define kAXMatteHoleAttribute               CFSTR("AXMatteHole")
#define kAXMatteContentUIElementAttribute CFSTR("AXMatteContentUIElement")


// Dock attributes
#define kAXIsApplicationRunningAttribute CFSTR("AXIsApplicationRunning")
```

```
// System-wide attributes
#define kAXFocusedApplicationAttribute      CFSTR("AXFocusedApplication")
```

Constants

`kAXRoleAttribute`

The role, or type, of this accessibility object (for example, `AXButton`). This string is for identification purposes only and does not need to be localized. All accessibility objects must include this attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSubroleAttribute`

The subrole of this accessibility object (for example, `AXCloseButton`). The subrole provides additional information about the accessibility object to an assistive application. This string is for identification purposes only and does not need to be localized. This attribute is necessary only for an accessibility object whose `AXRole` attribute does not adequately describe its meaning.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXRoleDescriptionAttribute`

A localized string describing the role (for example, "button"). This string must be readable by (or speakable to) the user. All accessibility objects must include this attribute. To get the system-defined role description string for a given role, use the `HICopyAccessibilityRoleDescription` (page 2077) function.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXHelpAttribute`

A localized string containing help text for this accessibility object. An accessibility object that provides help information should include this attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXTitleAttribute`

The title associated with this accessibility object. A title is text that the object displays as part of its visual interface, such as the text "OK" on an OK button. This string must be localizable and human-intelligible. This attribute is required for all accessibility objects that display a string in their visual interfaces.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXValueAttribute`

The value associated with this accessibility object (for example, a scroller value). The value of an accessibility object is user-modifiable and represents the setting of the associated user interface element, such as the contents of an editable text field or the position of a scroller. This attribute is required if an accessibility object's value state conveys information to the user or if the user can define the value of the object.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMinValueAttribute`

The minimum value this accessibility object can display (for example, the minimum value of a scroller control). This attribute is used only in conjunction with the `AXValue` attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMaxValueAttribute`

The maximum value this accessibility object can display (for example, the maximum value of a scroller control). This attribute is used only in conjunction with the `AXValue` attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXValueIncrementAttribute`

The amount an accessibility object's value changes as the result of a single action (for example, how far a scroller travels with one mouse click). This attribute is used only in conjunction with the `AXValue` attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXAllowedValuesAttribute`

An array of the allowed values for an accessibility object. This attribute indicates the subset of values to which an accessibility object can be set. For example, a slider control displays a large range of values, but the accessibility object representing the slider can be set to only a few specific values within that range. This attribute is used only in conjunction with the `AXValue` attribute.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXEnabledAttribute`

Indicates whether the user can interact with the accessibility object. For example, the `AXEnabled` attribute of a disabled button is `false`. This attribute is required for accessibility objects that represent views, menus, and menu items. This attribute is not required for accessibility objects that represent windows.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXFocusedAttribute`

Indicates whether the accessibility object currently has the keyboard focus. Note that you can set the value of the `AXFocused` attribute to `true` to accept keyboard focus. This attribute is required for all accessibility objects representing elements that can receive keyboard focus.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXParentAttribute`

This accessibility object's parent object in the accessibility hierarchy. This attribute is required for all accessibility objects except the application-level accessibility object.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXChildrenAttribute`

An array of the first-order accessibility objects contained by this accessibility object. An accessibility object may be a member of only one `AXChildren` array. This attribute is required for all accessibility objects that contain accessible child objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSelectedChildrenAttribute`

An array of selected first-order accessibility objects contained by this accessibility object. For example, the selected subelements of a list view are contained in the `AXSelectedChildren` array of the list view's accessibility object. The members of the `AXSelectedChildren` array are a subset of the members of this accessibility object's `AXChildren` array. This attribute is required for accessibility objects that contain selectable child objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXVisibleChildrenAttribute`

An array of first-order accessibility objects contained by this accessibility object that are visible to a sighted user. For example, a list view's `AXVisibleChildren` array would contain the list's subelements that are currently scrolled into view. The members of the `AXVisibleChildren` array are a subset of the members of this accessibility object's `AXChildren` array. This attribute is recommended for accessibility objects whose child objects can be scrolled out of view or otherwise obscured.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXWindowAttribute`

The window element that contains this accessibility object. An accessibility object that is contained in a window includes this attribute so an assistive application easily can find the window without having to step through all intervening objects in the accessibility hierarchy. Note that the value of the `AXWindow` attribute must be an accessibility object that represents a window, not a sheet or drawer. For a similar attribute that is less restrictive, see `kAXTopLevelUIElementAttribute`. The `AXWindow` attribute is required for all accessibility elements whose parent or more distant ancestor represents a window.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXPositionAttribute`

The global screen coordinates of the top-left corner of this accessibility object. Note that the coordinates `0,0` represent the top-left corner of the screen that displays the menu bar. All accessibility objects that have a screen position (in other words, are visible on the screen) should include this attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXTopLevelUIElementAttribute`

The window, sheet, or drawer element that contains this accessibility object. An accessibility object that is contained in a window, sheet, or drawer includes this attribute so an assistive application easily can find that element without having to step through all intervening objects in the accessibility hierarchy. This attribute is required for all accessibility objects whose parent or more distant ancestor represents a window, drawer, or sheet.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXSizeAttribute`

The vertical and horizontal dimensions of this accessibility object. This attribute is required for all accessibility objects that are visible on the screen.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXOrientationAttribute`

Indicates whether this accessibility object is displayed or interacted with in a vertical or a horizontal manner. The interpretation of an element, such as a slider, can change depending on whether it is oriented vertically or horizontally. Using the value of this attribute, an assistive application can communicate this information to the user. This attribute is required for any accessibility object, such as a scroller or slider, whose semantic meaning varies with the object's orientation.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXDescriptionAttribute`

The purpose of this accessibility object. The description string must be localizable and human-intelligible and it must be all lower case and include no punctuation. The string should briefly describe this accessibility object's purpose, without including the object's role description. This attribute is required for all accessibility objects that do not provide enough descriptive information in the title attribute.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXSelectedTextAttribute`

The currently selected text within this accessibility object. This attribute is required for all accessibility objects that represent editable text elements.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSelectedTextRangeAttribute`

Indicates the range of characters (not bytes) that defines the currently selected text within this accessibility object. This attribute is required for all accessibility objects that represent editable text elements.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXVisibleCharacterRangeAttribute`

Indicates the range of characters (not bytes) that are scrolled into view within this accessibility object. This attribute is required only for accessibility objects that represent an editable text area (objects of role `AXTextArea`), not for any other text-related accessibility objects.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXNumberOfCharactersAttribute`

The total number of characters (not bytes) in the editable text element represented by this accessibility object. This attribute is required for all accessibility objects that represent editable text elements.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXSharedTextUIElementsAttribute`

An array of accessibility objects with which the text of this accessibility object is shared. In a multi-column document, for example, each column may be represented by a separate accessibility object. However, the text in the document may flow from one column to the other. You get the value of this attribute if you need to know with which accessibility object this accessibility object shares its text. This attribute is recommended for sets of accessibility objects that share text in a single window. (See `kAXSharedCharacterRange` for a related attribute.)

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXSharedCharacterRangeAttribute`

The portion of shared text this accessibility object currently displays. In a multi-column document, for example, each column may be represented by a separate accessibility object. However, the text in the document may flow from one column to the other. Get the value of this attribute if you need to know the specific range of characters this accessibility object currently displays. This attribute is recommended for sets of accessibility objects that share text in a single window. (See `kAXSharedTextUIElementsAttribute` for a related attribute.)

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXMainAttribute`

Indicates whether the window represented by this accessibility object is the main application window. Note that a window can be main even though it does not have keyboard focus. This attribute is recommended for all accessibility objects that represent windows.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMinimizedAttribute`

Indicates whether the window represented by this accessibility object is currently minimized in the Dock. This attribute is recommended for all accessibility objects that represent windows that can be minimized.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXCloseButtonAttribute`

The close button of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's close button, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that contain a close button.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXZoomButtonAttribute`

The zoom button of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's zoom button, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that contain a zoom button.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMinimizeButtonAttribute`

The minimize button of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's minimize button, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that contain a minimize button.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXToolbarButtonAttribute`

The toolbar button of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's toolbar button, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that contain a toolbar button.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXGrowAreaAttribute`

The grow area of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's grow area, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that contain a grow area.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXProxyAttribute`

The document proxy of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's document proxy, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that display a document proxy.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXModalAttribute`

Indicates whether the window represented by this accessibility object is modal. This attribute is recommended for all accessibility objects that represent windows.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXDefaultButtonAttribute`

The default button of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's default button, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that contain a default button.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXCancelButtonAttribute`

The cancel button of the window represented by this accessibility object. An accessibility object includes this attribute to help an assistive application easily find a window's cancel button, without having to traverse the accessibility hierarchy. This attribute is recommended for all accessibility objects that represent windows that contain a cancel button.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXMenuItemCmdCharAttribute`

The primary key in the keyboard shortcut for the command represented by this accessibility object. For example, "O" is the primary key in the keyboard shortcut for the Open command.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMenuItemCmdVirtualKeyAttribute`

The key code associated with the physical key in the keyboard shortcut for the command represented by this accessibility object. For example, Return and Enter are different physical keys that can produce the same character. If an assistive application needs to be able to distinguish between them, it can view the virtual key codes.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMenuItemCmdGlyphAttribute`

The glyph displayed for a physical key in the keyboard shortcut for the command represented by this accessibility object, if it is different from the visible result of pressing the key. The Delete key, for example, produces an invisible character, but it is associated with a visible glyph.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMenuItemCmdModifiersAttribute`

An integer mask that represents the modifier keys held down in the keyboard shortcut for the command represented by this accessibility object.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMenuItemMarkCharAttribute`

The symbol displayed to the left of the menu item represented by this accessibility object. For example, in the Window menu, a checkmark appears next to the active document's name. For more information on the standard symbols that can appear next to menu items, see *Apple Human Interface Guidelines*.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMenuItemPrimaryUIElementAttribute`

The accessibility object representing the primary menu item in a group of dynamic menu items. Dynamic menu item are commands that change when the user presses a modifier key, such as Minimize Window and Minimize All Windows. Within each group, each dynamic menu item's accessibility object includes this attribute and in each case the attribute's value is the accessibility object representing the primary menu item.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXMenuBarAttribute`

The accessibility object representing the menu bar of this application. The application-level accessibility object includes this attribute to help an assistive application easily find the menu bar. This attribute is recommended for all application-level accessibility objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXWindowsAttribute`

An array of accessibility objects representing this application's windows. This attribute is recommended for all application-level accessibility objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXFrontmostAttribute`

Indicates whether the application represented by this accessibility object is active. This attribute is recommended for all application-level accessibility objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXHiddenAttribute`

Indicates whether the application represented by this accessibility object is hidden. This attribute is recommended for all application-level accessibility objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMainWindowAttribute`

The accessibility object representing this application's main window. This attribute is recommended for all application-level accessibility objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXFocusedWindowAttribute`

The accessibility object that represents the currently focused window of this application. This attribute is recommended for all application-level accessibility objects.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXFocusedUIElemenAttribute`

The accessibility object that represents the currently focused user interface element in this application. This attribute is recommended for all application-level accessibility objects.

`kAXHeaderAttribute`

The accessibility object representing the header element of this accessibility object. For example, a table or an outline view can have a header element that displays column or row headers. An accessibility object includes this attribute to help an assistive application easily find embedded header information. This attribute is recommended for all accessibility objects that represent elements that display header information.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXEditedAttribute`

Indicates whether the user interface element represented by this accessibility object has been edited. For example, a document window indicates it has been edited by displaying a black dot in its close button. This attribute is recommended for all accessibility objects that represent editable user interface elements.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXTitleUIElementAttribute`

An accessibility object that represents a static text title associated with another accessibility object.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXValueWrapsAttribute`

Indicates whether the value displayed in the user interface element represented by this accessibility object wraps around.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXTabsAttribute`

An array of accessibility objects representing the tabs this accessibility object displays. An accessibility object includes this attribute to help an assistive application easily distinguish a tab view's tabs from its other children. This attribute is recommended for all accessibility objects that represent tab views.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXHorizontalScrollBarAttribute`

The horizontal scroll bar displayed by the user interface element this accessibility object represents. This is a convenience attribute an assistive application can use easily to find the scroll bar without traversing the accessibility hierarchy. This attribute is recommended for all accessibility objects that display a horizontal scroll bar.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXVerticalScrollBarAttribute`

The vertical scroll bar displayed by the user interface element this accessibility object represents. This is a convenience attribute an assistive application can use easily to find the scroll bar without traversing the accessibility hierarchy. This attribute is recommended for all accessibility objects that display a vertical scroll bar.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXOverflowButtonAttribute`

Identifies which child of an accessibility object representing a toolbar is the overflow button (if any). This attribute is optional.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXFilenameAttribute`

The filename associated with this accessibility object. This attribute is optional.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXExpandedAttribute`

Indicates whether the menu displayed by the combo box or pop-up menu represented by this accessibility object is currently expanded. This attribute is recommended for all accessibility objects that display a pop-up menu.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSelectedAttribute`

Indicates whether the row or column element represented by this accessibility object is selected. This attribute is recommended for all accessibility objects that represent selectable rows or columns.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSplittersAttribute`

An array of views and splitter bar elements displayed by the split view represented by this accessibility object. This is a convenience attribute that helps an assistive application easily find these elements.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXNextContentsAttribute`

The group of accessibility objects representing the elements on one side of a splitter bar. (Which side of the splitter bar is considered "next" is determined by the value of the splitter bar's orientation attribute.) This attribute is recommended for an accessibility object that represents the splitter bar in a split view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXPreviousContentsAttribute`

The group of accessibility objects representing the elements on one side of a splitter bar. (Which side of the splitter bar is considered "previous" is determined by the value of the splitter bar's orientation attribute.) This attribute is recommended for an accessibility object that represents the splitter bar in a split view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXDocumentAttribute`

The URL of the open document represented by this accessibility object. This attribute represents the URL as a string object.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXIncrementButtonAttribute`

The increment element associated with the user interface object this accessibility object represents. This attribute can be used to provide convenient access to the increment area of a custom user interface object. To refer to the increment button associated with a date or time field, see `kAXIncrementorAttribute`.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXDecrementButtonAttribute`

The decrement element associated with the user interface object this accessibility object represents. This attribute can be used to provide convenient access to the decrement area of a custom user interface object.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXContentsAttribute`

Content-containing accessibility objects that are children of this accessibility object. For example, a tab view contains children that represent both the tab controls and the content displayed for each tab. The accessibility object representing a tab view can include only the content-display children in its `AXContents` attribute to help an assistive application provide more targeted information to the user. This attribute is recommended for any accessibility object whose children represent both content and control elements.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXIncrementorAttribute`

The incrementor of a time or date field represented by this accessibility object. This attribute is required for accessibility objects that represent time or date field elements that display an incrementor.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXHourFieldAttribute`

The hour field of a time field represented by this accessibility object. This attribute is required for accessibility objects that represent time fields that display hours.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXMinuteFieldAttribute`

The minute field of a time field represented by this accessibility object. This attribute is required for accessibility objects that represent time fields that display minutes.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXSecondFieldAttribute`

The second field of a time field represented by this accessibility object. This attribute is required for accessibility objects that represent time fields that display seconds.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXAMPMFieldAttribute`

The AM/PM field of a time field represented by this accessibility object. This attribute is required for accessibility objects that represent time fields that display AM/PM settings.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXDayFieldAttribute`

The day field of a time field represented by this accessibility object. This attribute is required for accessibility objects that represent time fields that display days.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXMonthFieldAttribute`

The month field of a time field represented by this accessibility object. This attribute is required for accessibility objects that represent time fields that display months.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXYearFieldAttribute`

The year field of a time field represented by this accessibility object. This attribute is required for accessibility objects that represent time fields that display years.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXColumnTitleAttribute`

The title of the column element represented by this accessibility object. Note that, because column titles are sometimes the children of a separate header element, the value of this attribute can refer to an element that is not a child of the column accessibility object.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXURLAttribute`

The URL that describes the location of the document or application represented by this accessibility object.

Available in Mac OS X v10.3 and later.

Declared in `AXAttributeConstants.h`.

`kAXLabelUIElementsAttribute`

An array of accessibility objects representing the labels displayed near the control represented by this accessibility object. For example, a slider control might display labels that indicate the range of values the slider can represent. Because these labels are not displayed as part of the slider's visual interface, an assistive application does not know they are associated with the slider. By including accessibility objects representing the labels in this attribute, you make this association explicit.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXLabelValueAttribute`

The value of the label represented by this accessibility object. This attribute is required for all accessibility objects that represent labels.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXShownMenuUIElementAttribute`

An array of accessibility objects that represent the contextual or Dock menus provided by this accessibility object.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXServesAsTitleForUIElementsAttribute`

An array of accessibility objects for which this accessibility object serves as the title. For example, a piece of static text can serve as a title for one or more user interface elements. Because this static text string is not displayed as part of any user interface element's visual interface, an assistive application does not know the title is associated with user interface elements. By including this attribute in the accessibility object representing the title, you specify the accessibility objects with which this title is associated.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXLinkedUIElementsAttribute`

An array of accessibility objects with which this accessibility object is related. For example, the contents of a list item can be displayed in another pane or window. The list item and the separately displayed contents are related, but this relationship may not be apparent to an assistive application. To make such a relationship explicit, you include this attribute in the accessibility objects representing the related user interface elements.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXRowsAttribute`

An array of the accessibility objects representing the rows in this table or outline view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXVisibleRowsAttribute`

An array of the accessibility objects representing the currently visible rows in this table or outline view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSelectedRowsAttribute`

An array of the accessibility objects representing the currently selected rows in this table or outline view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXColumnsAttribute`

An array of the accessibility objects representing the columns in this browser view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXVisibleColumnsAttribute`

An array of the accessibility objects representing the currently visible columns in this browser view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSelectedColumnsAttribute`

An array of the accessibility objects representing the currently selected columns in this browser view.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXSortDirectionAttribute`
The sort direction of this accessibility object's contents. For example, a list view's contents may be sorted in ascending or descending order.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXColumnHeaderUIElementsAttribute`
An array of accessibility objects representing the column headers of this table or browser view.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXIndexAttribute`
The index of the row or column represented by this accessibility object.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXDisclosingAttribute`
Indicates whether a row in an outline view represented by this accessibility object has an open or closed disclosure triangle. `true` indicates an open disclosure triangle; `false` indicates a closed disclosure triangle.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXDisclosedRowsAttribute`
An array of accessibility objects representing the disclosed rows of this user interface element.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXDisclosedByRowAttribute`
The accessibility object representing the disclosing row.

Available in Mac OS X v10.2 and later.

Declared in `AXAttributeConstants.h`.

`kAXMatteHoleAttribute`
The accessibility object that represents the area available to the user through the matte hole.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXMatteContentUIElementAttribute`
The accessibility object clipped by the matte.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXIsApplicationRunningAttribute`
Indicates if the application represented by the Dock icon this accessibility object represents is currently running.

Available in Mac OS X v10.4 and later.

Declared in `AXAttributeConstants.h`.

`kAXFocusedApplicationAttribute`
: Indicates the application element that is currently accepting keyboard input. This attribute is supported by the system-wide accessibility object to help an assistive application quickly determine the application that is accepting keyboard input. After the assistive application gets the accessibility object representing this application, it can send a message to the application asking for its focused accessibility object.

  Available in Mac OS X v10.2 and later.

  Declared in `AXAttributeConstants.h`.

`kAXInsertionPointLineNumberAttribute`
: The line number of the insertion point in the text associated with this accessibility object.

  Available in Mac OS X v10.4 and later.

  Declared in `AXAttributeConstants.h`.

Discussion
See the "Roles and Associated Attributes" appendix in Accessibility Overview for more information on which attributes are associated with a specific role.

Parameterized Attributes

Define the parameterized attributes an accessibility object can have.

```
// Text-suite parameterized attributes
#define kAXLineForIndexParameterizedAttribute CFSTR("AXLineForIndex")
#define kAXRangeForLineParameterizedAttribute          CFSTR("AXRangeForLine")
#define kAXStringForRangeParameterizedAttribute         CFSTR("AXStringForRange")
#define kAXRangeForPositionParameterizedAttribute CFSTR("AXRangeForPosition")
#define kAXRangeForIndexParameterizedAttribute CFSTR("AXRangeForIndex")
#define kAXBoundsForRangeParameterizedAttribute CFSTR("AXBoundsForRange")
#define kAXRTFForRangeParameterizedAttribute CFSTR("AXRTFForRange")
#define kAXAttributedStringForRangeParameterizedAttribute
CFSTR("AXAttributedStringForRange")
#define kAXStyleRangeForIndexParameterizedAttribute CFSTR("AXStyleRangeForIndex")
#define kAXInsertionPointLineNumberAttribute CFSTR("AXInsertionPointLineNumber")
```

Constants
`kAXLineForIndexParameterizedAttribute`
: Given an indexed character, the line number of the text associated with this accessibility object that contains the character.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

`kAXRangeForLineParameterizedAttribute`
: Given a line number, the range of characters of the text associated with this accessibility object that contains the line number.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

`kAXStringForRangeParameterizedAttribute`
: A substring of the text associated with this accessibility object that is specified by the given character range.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

`kAXRangeForPositionParameterizedAttribute`
: The composed character range in the text associated with this accessibility object that is specified by the given screen coordinates. This parameterized attribute returns the complete range of characters (including surrogate pairs of multi-byte glyphs) at the given screen coordinates.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

`kAXRangeForIndexParameterizedAttribute`
: The composed character range in the text associated with this accessibility object that is specified by the given index value. This parameterized attribute returns the complete range of characters (including surrogate pairs of multi-byte glyphs) at the given index.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

`kAXBoundsForRangeParameterizedAttribute`
: The bounding rectangle of the text associated with this accessibility object that is specified by the given range. This is the bounding rectangle a sighted user would see on the display screen, in pixels.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

`kAXRTFForRangeParameterizedAttribute`
: The RTF representation of the text associated with this accessibility object that is specified by the given range.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

`kAXAttributedStringForRangeParameterizedAttribute`
: The CFAttributedStringType representation of the text associated with this accessibility object that is specified by the given range.

  Available in Mac OS X v10.4 and later.

  Declared in `AXAttributeConstants.h`.

`kAXStyleRangeForIndexParameterizedAttribute`
: Given a character index, the range of text associated with this accessibility object over which the style in effect at that character index applies.

  Available in Mac OS X v10.3 and later.

  Declared in `AXAttributeConstants.h`.

Discussion
Parameterized attributes allow you to pass in additional values to get more specific information about the text associated with an accessibility object.

Availability
Available in Mac OS X version 10.3 and later.

Actions

Define the actions an accessibility object can perform.

```
// Accessibility actions.
#define kAXPressAction          CFSTR("AXPress")
#define kAXIncrementAction      CFSTR("AXIncrement")
#define kAXDecrementAction      CFSTR("AXDecrement")
#define kAXConfirmAction        CFSTR("AXConfirm")
#define kAXCancelAction         CFSTR("AXCancel")
#define kAXRaiseAction          CFSTR("AXRaise")
#define kAXShowMenuAction       CFSTR("AXShowMenu")
```

Constants

`kAXPressAction`

Simulates a single click, such as on a button.

Available in Mac OS X v10.2 and later.

Declared in `AXActionConstants.h`.

`kAXIncrementAction`

Increments the value of the accessibility object. The amount the value is incremented by is determined by the value of the `kAXValueIncrementAttribute` attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXActionConstants.h`.

`kAXDecrementAction`

Decrements the value of the accessibility object. The amount the value is decremented by is determined by the value of the `kAXValueIncrementAttribute` attribute.

Available in Mac OS X v10.2 and later.

Declared in `AXActionConstants.h`.

`kAXConfirmAction`

Simulates pressing the Return key.

Available in Mac OS X v10.2 and later.

Declared in `AXActionConstants.h`.

`kAXCancelAction`

Simulates pressing a Cancel button.

Available in Mac OS X v10.2 and later.

Declared in `AXActionConstants.h`.

`kAXRaiseAction`

Causes a window to become as frontmost as is allowed by the containing application's circumstances. Note that an application's floating windows (such as inspector windows) might remain above a window that performs the raise action.

Available in Mac OS X v10.3 and later.

Declared in `AXActionConstants.h`.

`kAXShowMenuAction`

Simulates the opening of a contextual menu in the element represented by this accessibility object. This action can also be used to simulate the display of a menu that is preassociated with an element, such as the menu that displays when a user clicks Safari's back button slowly.

Available in Mac OS X v10.4 and later.

Declared in `AXActionConstants.h`.

Notifications

Define the notifications that can be broadcast by an accessibility object.

```
// Focus notifications
#define kAXMainWindowChangedNotification CFSTR("AXMainWindowChanged")
#define kAXFocusedWindowChangedNotification CFSTR("AXFocusedWindowChanged")
#define kAXFocusedUIElementChangedNotification
CFSTR("AXFocusedUIElementChanged")

// Application notifications
#define kAXApplicationActivatedNotification CFSTR("AXApplicationActivated")
#define kAXApplicationDeactivatedNotification CFSTR("AXApplicationDeactivated")
#define kAXApplicationHiddenNotification CFSTR("AXApplicationHidden")
#define kAXApplicationShownNotification CFSTR("AXApplicationShown")

// Window notifications
#define kAXWindowCreatedNotification    CFSTR("AXWindowCreated")
#define kAXWindowMovedNotification      CFSTR("AXWindowMoved")
#define kAXWindowResizedNotification    CFSTR("AXWindowResized")
#define kAXWindowMiniaturizedNotification CFSTR("AXWindowMiniaturized")
#define kAXWindowDeminiaturizedNotification CFSTR("AXWindowDeminiaturized")

// New drawer, sheet, and help tag notifications
#define kAXDrawerCreatedNotification    CFSTR("AXDrawerCreated")
#define kAXSheetCreatedNotification     CFSTR("AXSheetCreated")
#define kAXHelpTagCreatedNotification   CFSTR("AXHelpTagCreated")

// Element notifications
#define kAXValueChangedNotification     CFSTR("AXValueChanged")
#define kAXUIElementDestroyedNotification       CFSTR("AXUIElementDestroyed")

// Menu notifications
#define kAXMenuOpenedNotification       CFSTR("AXMenuOpened")
#define kAXMenuClosedNotification       CFSTR("AXMenuClosed")
#define kAXMenuItemSelectedNotification CFSTR("AXMenuItemSelected")

// Table and outline view notifications
#define kAXRowCountChangedNotification  CFSTR("AXRowCountChanged")

// Miscellaneous notifications
#define kAXSelectedChildrenChangedNotification
CFSTR("AXSelectedChildrenChanged")
#define kAXResizedNotification          CFSTR("AXResized")
#define kAXMovedNotification            CFSTR("AXMoved")
#define kAXCreatedNotification          CFSTR("AXCreated")
```

Constants

`kAXMainWindowChangedNotification`

The main window has changed.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXFocusedWindowChangedNotification`

The focused window has changed.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXFocusedUIElementChangedNotification`
The focused accessibility object has changed.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXApplicationActivatedNotification`
The application was activated (that is, brought to front).

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXApplicationDeactivatedNotification`
The application was deactivated.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXApplicationHiddenNotification`
The application was hidden.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXApplicationShownNotification`
The application was shown (that is, a hidden application is now visible).

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXWindowCreatedNotification`
A window was created. Carbon automatically sends this notification when window is created, as long as the window is implemented using Carbon window mechanisms.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXWindowMovedNotification`
The window was moved (this notification is sent at the end of the window-move operation, not during it).

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXWindowResizedNotification`
The window was resized (this notification is sent at the end of the window-resize operation, not during it).

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXWindowMiniaturizedNotification`
The application was minimized (that is, moved into the Dock).

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXWindowDeminiaturizedNotification`
The window was moved out of the Dock.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXDrawerCreatedNotification`
A drawer was created (that is, a drawer now extends from this window).

Available in Mac OS X v10.3 and later.

Declared in `AXNotificationConstants.h`.

`kAXSheetCreatedNotification`
A sheet was created (that is, a modal dialog now extends from this window).

Available in Mac OS X v10.3 and later.

Declared in `AXNotificationConstants.h`.

`kAXHelpTagCreatedNotification`
A help tag is now visible for this accessibility object.

Available in Mac OS X v10.4 and later.

Declared in `AXNotificationConstants.h`.

`kAXValueChangedNotification`
The value of an accessibility object's value attribute was changed.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXUIElementDestroyedNotification`
An accessibility object was disposed of.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXMenuOpenedNotification`
A menu was opened.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXMenuClosedNotification`
A menu was closed.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXMenuItemSelectedNotification`
A menu item was selected.

Available in Mac OS X v10.2 and later.

Declared in `AXNotificationConstants.h`.

`kAXRowCountChangedNotification`
The number of rows in this table was changed.

Available in Mac OS X v10.4 and later.

Declared in `AXNotificationConstants.h`.

`kAXSelectedChildrenChangedNotification`
A different subset of this accessibility object's children were selected.

Available in Mac OS X v10.4 and later.

Declared in `AXNotificationConstants.h`.

`kAXResizedNotification`
The window has changed size.

Available in Mac OS X v10.4 and later.

Declared in `AXNotificationConstants.h`.

`kAXMovedNotification`
The position of this accessibility object was changed.

Available in Mac OS X v10.4 and later.

Declared in `AXNotificationConstants.h`.

`kAXCreatedNotification`
An accessibility object was created.

Available in Mac OS X v10.4 and later.

Declared in `AXNotificationConstants.h`.

Orientations and Sort Directions

Define the values for the orientation and sort-direction attributes of some accessibility objects.

```
// Orientations
#define kAXHorizontalOrientationValue CFSTR("AXHorizontalOrientation")
#define kAXVerticalOrientationValue CFSTR("AXVerticalOrientation")
#define kAXUnknownOrientationValue  CFSTR("AXUnknownOrientation")

// Sort directions
#define kAXAscendingSortDirectionValue CFSTR("AXAscendingSortDirection")
#define kAXDescendingSortDirectionValue CFSTR("AXDescendingSortDirection")
#define kAXUnknownSortDirectionValue CFSTR("AXUnknownSortDirection")
```

Constants

`kAXHorizontalOrientationValue`
This object is oriented horizontally.

Available in Mac OS X v10.2 and later.

Declared in `AXValueConstants.h`.

`kAXVerticalOrientationValue`
This object is oriented vertically.

Available in Mac OS X v10.2 and later.

Declared in `AXValueConstants.h`.

`kAXUnknownOrientationValue`
The orientation of this object is unknown.

Available in Mac OS X v10.4 and later.

Declared in `AXValueConstants.h`.

`kAXAscendingSortDirectionValue`
This object's contents are sorted in ascending order.

Available in Mac OS X v10.4 and later.

Declared in `AXValueConstants.h`.

`AXDescendingSortDirection`
This object's contents are sorted in descending order.

`kAXUnknownSortDirectionValue`
The sort order of this object is unknown.

Available in Mac OS X v10.4 and later.

Declared in `AXValueConstants.h`.

Discussion
See `kAXOrientationAttribute` and `kAXSortDirectionAttribute` for more information on the attributes for which you can use these values.

Result Codes

The result codes returned by the Carbon accessibility implementation are listed below. Other result codes defined in `AXError.h` are of use only to assistive applications.

Result Code	Value	Description
`kAXErrorIllegalArgument`	-25201	The value received in this event is an invalid value for this attribute. This also applies for invalid parameters in parameterized attributes. Available in Mac OS X v10.2 and later.
`kAXErrorInvalidUIElement`	-25202	The accessibility object received in this event is invalid. Available in Mac OS X v10.2 and later.
`kAXErrorInvalidUIElementObserver`	-25203	The observer for the accessibility object received in this event is invalid. Available in Mac OS X v10.2 and later.
`kAXErrorCannotComplete`	-25204	A fundamental error has occurred, such as a failure to allocate memory during processing. Available in Mac OS X v10.2 and later.
`kAXErrorAttributeUnsupported`	-25205	The referenced attribute is not supported. Alternatively, you can return the `eventNotHandledErr` error. Available in Mac OS X v10.2 and later.
`kAXErrorActionUnsupported`	-25206	The referenced action is not supported. Alternatively, you can return the `eventNotHandledErr` error. Available in Mac OS X v10.2 and later.
`kAXErrorAPIDisabled`	-25211	Assistive applications are not enabled in System Preferences. Available in Mac OS X v10.2 and later.

Result Code	Value	Description
`kAXErrorParameterizedAttributeUnsupported`	-25213	The parameterized attribute is not supported. Alternatively, you can return the `eventNotHandledErr` error. Available in Mac OS X v10.3 and later.

Core Printing Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	PMCore.h PMCoreDeprecated.h PMDefinitions.h PMDefinitionsDeprecated.h

Overview

Core Printing is a C API that Mac OS X applications and command line tools can use to perform printing tasks that don't display a user interface. Core Printing defines a set of opaque types and a rich set of operations on instances of these types. The Core Printing opaque types include:

- `PMPrintSession` for general information about a print job
- `PMPrintSettings` for print job parameters
- `PMPageFormat` for the page format of a printed document
- `PMPaper` for information about a type of paper
- `PMPrinter` for information about a printer

In Carbon applications, Core Printing is used together with Carbon Printing to implement printing features. For more information about Carbon Printing, see *Carbon Printing Reference*.

In Cocoa applications, Core Printing can be used to extend the functionality in the Cocoa printing classes. The `NSPrintInfo` class provides direct access to some Core Printing objects.

Note: Core Printing is available to 64-bit applications, except for functions, data types, and constants that have been deprecated.

Functions by Task

Releasing and Retaining Printing Objects

`PMRelease` (page 2214)
Releases a printing object by decrementing its reference count.

`PMRetain` (page 2215)
Retains a printing object by incrementing its reference count.

Creating and Using Page Format Objects

`PMCreatePageFormat` (page 2143)
Creates a new page format object.

`PMCreatePageFormatWithPMPaper` (page 2144)
Creates a page format object with a specified paper.

`PMCopyPageFormat` (page 2141)
Copies the settings from one page format object into another.

`PMSessionDefaultPageFormat` (page 2225)
Assigns default parameter values to a page format object used in the specified printing session.

`PMSessionValidatePageFormat` (page 2248)
Updates the values in a page format object and validates them against the current formatting printer.

`PMSessionCreatePageFormatList` (page 2223)
Obtains a list of page format objects, each of which describes a paper size available on the specified printer.

`PMPageFormatCreateDataRepresentation` (page 2174)
Creates a data representation of a page format object.

`PMPageFormatCreateWithDataRepresentation` (page 2175)
Creates a page format object from a data representation.

`PMFlattenPageFormatToCFData` (page 2149)
Flattens a page format object into a Core Foundation data object for storage in a user document. (Deprecated. Use `PMPageFormatCreateDataRepresentation` (page 2174) instead.)

`PMFlattenPageFormat` (page 2149) Deprecated in Mac OS X v10.5
Flattens a page format object into a Memory Manager handle for storage in a user document. (Deprecated. Use `PMPageFormatCreateDataRepresentation` (page 2174) instead.)

`PMFlattenPageFormatToURL` (page 2150) Deprecated in Mac OS X v10.5
Flattens a page format object into a file for storage in a user document. (Deprecated. Use `PMPageFormatCreateDataRepresentation` (page 2174) and write the resulting data to your destination.)

`PMUnflattenPageFormat` (page 2265) Deprecated in Mac OS X v10.5
Rebuilds a page format object from a Memory Manager handle that contains flattened page format data. (Deprecated. Use `PMPageFormatCreateWithDataRepresentation` (page 2175) instead.)

`PMUnflattenPageFormatWithCFData` (page 2266) Deprecated in Mac OS X v10.5
Rebuilds a page format object from a Core Foundation data object that contains flattened page format data. (Deprecated. Use `PMPageFormatCreateWithDataRepresentation` (page 2175) instead.)

`PMUnflattenPageFormatWithURL` (page 2266) Deprecated in Mac OS X v10.5
Rebuilds a page format object from a file system URL that contains flattened page format data. (Deprecated. Instead read the data into a CFData object and use `PMPageFormatCreateWithDataRepresentation` (page 2175).)

Accessing Data in Page Format Objects

`PMGetPageFormatExtendedData` (page 2164)
Obtains extended page format data previously stored by your application.

`PMSetPageFormatExtendedData` (page 2258)
Stores your application-specific data in a page format object.

`PMGetPageFormatPaper` (page 2164)
Obtains the paper associated with a page format object.

`PMPageFormatGetPrinterID` (page 2175)
Obtains the identifier of the formatting printer for a page format object.

`PMGetOrientation` (page 2163)
Obtains the current setting for page orientation.

`PMSetOrientation` (page 2257)
Sets the page orientation for printing.

`PMGetScale` (page 2170)
Obtains the scaling factor currently applied to the page and paper rectangles.

`PMSetScale` (page 2263)
Sets the scaling factor for the page and paper rectangles.

`PMGetAdjustedPageRect` (page 2153)
Obtains the imageable area or page rectangle, taking into account orientation, application drawing resolution, and scaling settings.

`PMGetAdjustedPaperRect` (page 2154)
Obtains the rectangle defining the paper size, taking into account orientation, application drawing resolution, and scaling settings.

`PMGetUnadjustedPageRect` (page 2170)
Obtains the imageable area or page rectangle, unaffected by orientation, resolution, or scaling.

`PMGetUnadjustedPaperRect` (page 2171)
Obtains the paper rectangle, unaffected by rotation, resolution, or scaling.

`PMGetResolution` (page 2169) Deprecated in Mac OS X v10.5
Obtains the current application's drawing resolution. (Deprecated. Draw using Quartz 2D and call `CGContextScaleCTM` (page 105) instead.)

`PMSetAdjustedPageRect` (page 2250) Deprecated in Mac OS X v10.5
Requests a particular page size, adjusted for the current rotation, resolution, or scaling settings. (Deprecated. To set a particular paper size and margins, obtain or create a `PMPaper` (page 2275) object and call `PMCreatePageFormatWithPMPaper` (page 2144).)

`PMSetResolution` (page 2263) Deprecated in Mac OS X v10.5
Sets the application drawing resolution. (Deprecated. Draw using Quartz 2D and call `CGContextScaleCTM` (page 105) instead.)

`PMSetUnadjustedPaperRect` (page 2264) Deprecated in Mac OS X v10.5
Requests a particular paper size, unaffected by rotation, resolution, or scaling. (Deprecated. To set a particular paper size, obtain or create a `PMPaper` (page 2275) object and call `PMCreatePageFormatWithPMPaper` (page 2144).)

Creating and Using Print Settings Objects

`PMCreatePrintSettings` (page 2144)
Creates a new print settings object.

`PMSessionDefaultPrintSettings` (page 2225)
Assigns default parameter values to a print settings object for the specified printing session.

PMSessionValidatePrintSettings (page 2249)
Validates a print settings object within the context of the specified printing session.

PMPrintSettingsCreateDataRepresentation (page 2208)
Creates a data representation of a print settings object.

PMPrintSettingsCreateWithDataRepresentation (page 2209)
Creates a print settings object from a data representation.

PMCopyPrintSettings (page 2142)
Copies the settings from one print settings object into another.

PMPrintSettingsToOptions (page 2213)
Converts print settings into a CUPS options string.

PMPrintSettingsToOptionsWithPrinterAndPageFormat (page 2213)
Converts print settings and page format data into a CUPS options string for a specified printer.

PMFlattenPrintSettings (page 2150) Deprecated in Mac OS X v10.5
Flattens a print settings object into a Memory Manager handle for storage in a user document. (Deprecated. Use PMPrintSettingsCreateDataRepresentation (page 2208) instead.)

PMFlattenPrintSettingsToCFData (page 2151) Deprecated in Mac OS X v10.5
Flattens a print settings object into a Core Foundation data object for storage in a user document. (Deprecated. Use PMPrintSettingsCreateDataRepresentation (page 2208) instead.)

PMFlattenPrintSettingsToURL (page 2152) Deprecated in Mac OS X v10.5
Flattens a print settings object into a URL for storage in a user document. (Deprecated. Instead use PMPrintSettingsCreateDataRepresentation (page 2208) and write the resulting data to your destination.)

PMUnflattenPrintSettings (page 2267) Deprecated in Mac OS X v10.5
Rebuilds a print settings object from a Memory Manager handle that contains flattened print settings data. (Deprecated. Use PMPrintSettingsCreateWithDataRepresentation (page 2209) instead.)

PMUnflattenPrintSettingsWithCFData (page 2268) Deprecated in Mac OS X v10.5
Rebuilds a print settings object from a Core Foundation data object that contains flattened print settings data. (Deprecated. Use PMPrintSettingsCreateWithDataRepresentation (page 2209) instead.)

PMUnflattenPrintSettingsWithURL (page 2268) Deprecated in Mac OS X v10.5
Rebuilds a print settings object from a file that contains flattened print settings data. (Deprecated. Instead read the data into a CFData object and use PMPrintSettingsCreateWithDataRepresentation (page 2209).)

Accessing Data in Print Settings Objects

PMGetPrintSettingsExtendedData (page 2168)
Obtains extended print settings data previously stored by your application.

PMSetPrintSettingsExtendedData (page 2261)
Stores your application-specific data in a print settings object.

PMGetFirstPage (page 2159)
Obtains the number of the first page to be printed.

PMSetFirstPage (page 2254)
Sets the default page number of the first page to be printed.

`PMGetLastPage` (page 2162)
Obtains the number of the last page to be printed.

`PMSetLastPage` (page 2257)
Sets the page number of the last page to be printed.

`PMGetPageRange` (page 2165)
Obtains the valid range of pages that can be printed.

`PMSetPageRange` (page 2259)
Sets the valid range of pages that can be printed.

`PMPrintSettingsGetJobName` (page 2210)
Obtains the name of a print job.

`PMPrintSettingsSetJobName` (page 2211)
Specifies the name of a print job.

`PMGetCopies` (page 2155)
Obtains the number of copies that the user requests to be printed.

`PMSetCopies` (page 2252)
Sets the initial value for the number of copies to be printed.

`PMGetCollate` (page 2154)
Obtains a Boolean value that indicates whether the job collate option is selected.

`PMSetCollate` (page 2251)
Specifies whether the job collate option is selected.

`PMGetDuplex` (page 2158)
Obtains the selected duplex mode.

`PMSetDuplex` (page 2253)
Sets the duplex mode.

`PMPrintSettingsGetValue` (page 2210)
Obtains the value of a setting in a print settings object.

`PMPrintSettingsSetValue` (page 2212)
Stores the value of a setting in a print settings object.

`PMPrintSettingsCopyAsDictionary` (page 2207)
Creates a dictionary that contains the settings in a print settings object.

`PMPrintSettingsCopyKeys` (page 2207)
Obtains the keys for items in a print settings object.

`PMGetJobNameCFString` (page 2161) Deprecated in Mac OS X v10.5
Obtains the name of the print job. (Deprecated. Use `PMPrintSettingsGetJobName` (page 2210) instead.)

`PMSetJobNameCFString` (page 2256) Deprecated in Mac OS X v10.5
Specifies the name of a print job. (Deprecated. Use `PMPrintSettingsSetJobName` (page 2211) instead.)

Creating Printing Session Objects

`PMCreateSession` (page 2145)
Creates and initializes a printing session object and creates a context for printing operations.

Accessing Data in Printing Session Objects

`PMSessionGetDataFromSession` (page 2231)
Obtains application-specific data previously stored in a printing session object.

`PMSessionSetDataInSession` (page 2242)
Stores your application-specific data in a printing session object.

`PMSessionGetCurrentPrinter` (page 2231)
Obtains the current printer associated with a printing session.

`PMSessionSetCurrentPMPrinter` (page 2241)
Changes the current printer for a printing session.

`PMSessionGetCGGraphicsContext` (page 2230)
Obtains the Quartz graphics context for the current page in a printing session.

`PMSessionError` (page 2229)
Obtains the result code for any error returned by the printing session.

`PMSessionSetError` (page 2246)
Sets the value of the current result code for the specified printing session.

`PMSessionGetGraphicsContext` (page 2234) Deprecated in Mac OS X v10.5
Obtains the graphics context for the current page in a printing session. (Deprecated. Use `PMSessionGetCGGraphicsContext` (page 2230) instead.)

`PMSessionGeneral` (page 2229) Deprecated in Mac OS X v10.4
Maintains compatibility with the `PrGeneral` function in the classic Printing Manager. (Deprecated. Use `PMPrinterGetCommInfo` (page 2190) instead.)

`PMSessionGetDocumentFormatGeneration` (page 2233) Deprecated in Mac OS X v10.4
Obtains the spool file formats that can be generated for the specified printing session. (Deprecated. If you're drawing using Quartz 2D instead of QuickDraw, use `PMSessionBeginCGDocument` or `PMSessionBeginCGDocumentNoDialog` (page 2217); for submitting PostScript data, use `PMPrinterPrintWithFile` (page 2203) or `PMPrinterPrintWithProvider` (page 2204); to draw EPS data, use `PMCGImageCreateWithEPSDataProvider` (page 2139).)

`PMSessionSetCurrentPrinter` (page 2242) Deprecated in Mac OS X v10.4
Changes the current printer for a printing session to a printer specified by name. (Deprecated. Use `PMSessionSetCurrentPMPrinter` (page 2241) instead.)

`PMSessionSetDocumentFormatGeneration` (page 2244) Deprecated in Mac OS X v10.4
Requests a specified spool file format and supplies the graphics context type to use for drawing pages within the print loop. (Deprecated. If you're drawing using Quartz 2D instead of QuickDraw, use `PMSessionBeginCGDocument` or `PMSessionBeginCGDocumentNoDialog` (page 2217); for submitting PostScript data, use `PMPrinterPrintWithFile` (page 2203) or `PMPrinterPrintWithProvider` (page 2204); to draw EPS data, use `PMCGImageCreateWithEPSDataProvider` (page 2139).)

Using Printer Presets

`PMPresetCopyName` (page 2185)
Obtains the localized name for a preset.

`PMPresetCreatePrintSettings` (page 2186)
Creates a print settings object with settings that correspond to a preset.

`PMPresetGetAttributes` (page 2187)
Obtains the attributes of a preset.

Creating and Using Paper Objects

`PMPaperCreate` (page 2176)
: Creates a paper object. (Deprecated. Use `PMPrinterGetPaperList` (page 2198) to find the built-in papers available for a given printer or use `PMPaperCreateCustom` (page 2177) to create a custom paper.)

`PMPaperCreateCustom` (page 2177)
: Creates a custom paper object.

`PMPaperIsCustom` (page 2183)
: Returns a Boolean value indicating whether a specified paper is a custom paper.

Accessing Data in Paper Objects

`PMPaperGetID` (page 2179)
: Obtains the identifier of a paper object.

`PMPaperGetName` (page 2180)
: Obtains the name for a given paper.

`PMPaperGetWidth` (page 2182)
: Obtains the width of the sheet of paper represented by a paper object.

`PMPaperGetHeight` (page 2179)
: Obtains the height of the sheet of paper represented by a paper object.

`PMPaperGetMargins` (page 2180)
: Obtains the margins describing the unprintable area of the sheet represented by a paper object.

`PMPaperCreateLocalizedName` (page 2178)
: Obtains the localized name for a given paper.

`PMPaperGetPrinterID` (page 2182)
: Obtains the printer ID of the printer to which a given paper corresponds.

`PMPaperGetPPDPaperName` (page 2181)
: Obtains the PPD paper name for a given paper.

Print Loop Functions

`PMSessionBeginCGDocumentNoDialog` (page 2217)
: Begins a print job that draws into a Quartz graphics context and suppresses the printing status dialog.

`PMSessionEndDocumentNoDialog` (page 2227)
: Ends a print job started by calling the function `PMSessionBeginCGDocumentNoDialog` (page 2217) or `PMSessionBeginDocumentNoDialog` (page 2218).

`PMSessionBeginPageNoDialog` (page 2219)
: Starts a new page for printing in the specified printing session and suppresses the printing status dialog.

`PMSessionEndPageNoDialog` (page 2228)
: Indicates the end of drawing the current page for the specified printing session.

PMSessionBeginDocumentNoDialog (page 2218) Deprecated in Mac OS X v10.5
Begins a print job that, by default, draws into a QuickDraw graphics port, and suppresses the printing status dialog. (Deprecated. Use PMSessionBeginCGDocumentNoDialog (page 2217) instead.)

PMSessionSetIdleProc (page 2247) Deprecated in Mac OS X v10.4
Installs an idle callback function in your print loop. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

Accessing the Print Job Destination

PMSessionSetDestination (page 2243)
Sets the destination location, format, and type for a print job.

PMSessionGetDestinationType (page 2232)
Obtains the output destination for a print job.

PMSessionCopyDestinationFormat (page 2221)
Obtains the destination format for a print job.

PMSessionCopyDestinationLocation (page 2221)
Obtains a destination location for a print job.

PMSessionCopyOutputFormatList (page 2222)
Obtains an array of destination formats supported by the current print destination.

Creating Printer Objects

PMServerLaunchPrinterBrowser (page 2216)
Launches the printer browser to browse the printers available for a print server.

PMServerCreatePrinterList (page 2215)
Creates a list of printers available to a print server.

PMSessionCreatePrinterList (page 2224)
Creates a list of printers available in the specified printing session.

PMPrinterCreateFromPrinterID (page 2190)
Creates a printer object from a print queue identifier.

PMCreateGenericPrinter (page 2143)
Creates a generic printer object.

Accessing Information About a Printer

PMPrinterCopyDescriptionURL (page 2187)
Obtains the URL of the description file for a given printer.

PMPrinterCopyDeviceURI (page 2188)
Obtains the device URI of a given printer.

PMPrinterCopyHostName (page 2188)
Obtains the name of the server hosting the print queue for a given printer.

PMPrinterCopyPresets (page 2189)
Obtains a list of print settings presets for a printer.

PMPrinterGetCommInfo (page 2190)
Obtains information about the communication channel for a printer.

PMPrinterGetDeviceURI (page 2192)
Obtains a copy of a printer's device URI. (Deprecated. Use PMPrinterCopyDeviceURI (page 2188) instead.)

PMPrinterGetDriverCreator (page 2192)
Obtains the creator of the driver associated with the specified printer.

PMPrinterGetDriverReleaseInfo (page 2193)
Obtains version information for the driver associated with the specified printer.

PMPrinterGetID (page 2193)
Returns the unique identifier of a printer.

PMPrinterGetLanguageInfo (page 2194)
Obtains information about the imaging language for the specified printer.

PMPrinterGetLocation (page 2195)
Returns the location of a printer.

PMPrinterGetMakeAndModelName (page 2195)
Obtains the manufacturer and model name of the specified printer.

PMPrinterGetMimeTypes (page 2196)
Obtains a list of MIME content types supported by a printer using the specified print settings.

PMPrinterGetName (page 2197)
Returns the human-readable name of a printer.

PMPrinterGetOutputResolution (page 2197)
Obtains the printer hardware output resolution for the specified print settings.

PMPrinterSetOutputResolution (page 2205)
Sets the print settings to reflect the specified printer hardware output resolution.

PMPrinterGetPaperList (page 2198)
Obtains the list of papers available for a printer.

PMPrinterGetPrinterResolutionCount (page 2199)
Obtains the number of resolution settings supported by the specified printer.

PMPrinterGetIndexedPrinterResolution (page 2194)
Obtains a resolution setting based on an index into the range of settings supported by the specified printer.

PMPrinterGetState (page 2200)
Obtains the current state of the print queue for a printer.

PMPrinterSetDefault (page 2205)
Sets the default printer for the current user.

PMPrinterIsDefault (page 2200)
Returns a Boolean value indicating whether a printer is the default printer for the current user.

PMPrinterIsFavorite (page 2201)
Returns a Boolean value indicating whether a printer is in the user's list of favorite printers.

PMPrinterIsPostScriptCapable (page 2201)
Returns a Boolean value indicating whether a printer is PostScript capable.

PMPrinterIsPostScriptPrinter (page 2202)
Determines whether a printer is a PostScript printer.

PMPrinterIsRemote (page 2202)
Indicates whether a printer is hosted by a remote print server.

PMPrinterGetPrinterResolution (page 2199) Deprecated in Mac OS X v10.5
Obtains a resolution setting for the specified printer. (Deprecated. Use PMPrinterGetPrinterResolutionCount (page 2199) and PMPrinterGetIndexedPrinterResolution (page 2194) to examine the available printer resolutions.)

PMPrinterGetDescriptionURL (page 2191) Deprecated in Mac OS X v10.4
Obtains a reference to the specified printer's description file. (Deprecated. Use PMPrinterCopyDescriptionURL (page 2187) instead.)

Submitting a Print Job to a Printer

PMPrinterPrintWithFile (page 2203)
Submits a print job to a specified printer using a file that contains print data.

PMPrinterPrintWithProvider (page 2204)
Submits a print job to a specified printer using a Quartz data provider to obtain the print data.

Accessing PostScript Printer Description Files

PMCopyAvailablePPDs (page 2140)
Obtains the list of PostScript printer description (PPD) files in a PPD domain.

PMCopyLocalizedPPD (page 2140)
Obtains a localized PostScript printer description (PPD) file.

PMCopyPPDData (page 2142)
Obtains the uncompressed PPD data for a PostScript printer description (PPD) file.

Printing with PostScript Data

PMCGImageCreateWithEPSDataProvider (page 2139)
Creates an image that references both the PostScript contents of EPS data and a preview (proxy) image for the data.

PMPrinterWritePostScriptToURL (page 2206)
Converts an input file of the specified MIME type to printer-ready PostScript for a destination printer.

PMSessionPostScriptBegin (page 2237) Deprecated in Mac OS X v10.4
Puts the current printer driver into PostScript mode, ready to accept PostScript data instead of QuickDraw data. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSessionPostScriptData (page 2238) Deprecated in Mac OS X v10.4
Passes PostScript data, referenced by a pointer, to the current printer driver. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSessionPostScriptEnd (page 2239) Deprecated in Mac OS X v10.4
: Restores the current driver to QuickDraw mode, ready to accept QuickDraw data instead of PostScript data. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSessionPostScriptFile (page 2239) Deprecated in Mac OS X v10.4
: Passes the PostScript data, contained in a file, to the current printer driver. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSessionPostScriptHandle (page 2240) Deprecated in Mac OS X v10.4
: Passes the PostScript data, referenced by a Memory Manager handle, to the current printer driver. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSessionSetPSInjectionData (page 2247) Deprecated in Mac OS X v10.4
: Specifies a set of PostScript code injection points and the PostScript data to be injected. (Deprecated. Use PMPrinterPrintWithFile (page 2203) or PMPrinterPrintWithProvider (page 2204) instead.)

Using PDF Workflow Items

PMWorkflowCopyItems (page 2270)
: Obtains an array of the available PDF workflow items.

PMWorkflowSubmitPDFWithOptions (page 2271)
: Submits a PDF file for workflow processing using the specified CUPS options string.

PMWorkflowSubmitPDFWithSettings (page 2272)
: Submits a PDF file for workflow processing using the specified print settings.

Matching Color With ColorSync

PMSessionDisableColorSync (page 2226) Deprecated in Mac OS X v10.5
: Disables use of a custom ColorSync profile previously enabled by the function PMSessionEnableColorSync (page 2226). (Deprecated. There is no replacement; draw using Quartz 2D instead.)

PMSessionEnableColorSync (page 2226) Deprecated in Mac OS X v10.5
: Enables use of a custom ColorSync profile previously set by the function PMSetProfile (page 2262). (Deprecated. There is no replacement; draw using Quartz 2D instead.)

PMSetProfile (page 2262) Deprecated in Mac OS X v10.5
: Embeds a color profile during printing. (Deprecated. There is no replacement; draw using Quartz 2D instead.)

Converting and Saving Old Print Records

PMSessionConvertOldPrintRecord (page 2220) Deprecated in Mac OS X v10.4
: Creates new page format and print settings objects from an old-style print record created for the classic Printing Manager. (Deprecated. There is no replacement; during the transition from Mac OS 9 to Mac OS X, this function facilitated the migration of print records saved in documents created in Mac OS 9, but the function no longer serves any useful purpose in Mac OS X.)

`PMSessionMakeOldPrintRecord` (page 2236) Deprecated in Mac OS X v10.4
Creates an old-style print record from page format and print settings objects. (Deprecated. There is no replacement; old-style print records are obsolete and serve no useful purpose in Mac OS X.)

Creating, Calling, and Deleting Universal Procedure Pointers

`DisposePMIdleUPP` (page 2137) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to an idle callback. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

`InvokePMIdleUPP` (page 2137) Deprecated in Mac OS X v10.4
Calls an idle callback. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

`NewPMIdleUPP` (page 2138) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to an idle callback. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

Legacy Core Printing Functions

`PMGetDestination` (page 2156) Deprecated in Mac OS X v10.5
Obtains the output destination of a print job. (Deprecated. Use `PMSessionGetDestinationType` (page 2232), `PMSessionCopyDestinationFormat` (page 2221), or `PMSessionCopyDestinationLocation` (page 2221) instead.)

`PMBegin` (page 2138) Deprecated in Mac OS X v10.4
Prepares Core Printing for use. (Deprecated. Use `PMCreateSession` (page 2145) instead.)

`PMConvertOldPrintRecord` (page 2139) Deprecated in Mac OS X v10.4
Creates a new `PMPageFormat` object and a new `PMPrintSettings` object from a print record created by the classic Printing Manager. (Deprecated. There is no replacement; during the transition from Mac OS 9 to Mac OS X, this function facilitated the migration of print records saved in documents created in Mac OS 9, but the function no longer serves any useful purpose in Mac OS X.)

`PMDefaultPageFormat` (page 2145) Deprecated in Mac OS X v10.4
Assigns default parameter values to an existing `PMPageFormat` object, for the current printer. (Deprecated. Use `PMSessionDefaultPageFormat` (page 2225) instead.)

`PMDefaultPrintSettings` (page 2146) Deprecated in Mac OS X v10.4
Assigns default parameter values to a `PMPrintSettings` object. (Deprecated. Use `PMSessionDefaultPrintSettings` (page 2225) instead.)

`PMDisableColorSync` (page 2146) Deprecated in Mac OS X v10.4
Disables ColorSync color matching for the current page. (Deprecated. There is no replacement; draw using Quartz 2D instead.)

`PMDisposePageFormat` (page 2147) Deprecated in Mac OS X v10.4
Releases memory previously allocated for a `PMPageFormat` object. (Deprecated. Use `PMRelease` (page 2214) instead.)

`PMDisposePrintSettings` (page 2147) Deprecated in Mac OS X v10.4
Releases memory previously allocated for a `PMPrintSettings` object. (Deprecated. Use `PMRelease` (page 2214) instead.)

PMEnableColorSync (page 2148) Deprecated in Mac OS X v10.4
Enables ColorSync color matching for the current page. (Deprecated. There is no replacement; draw using Quartz 2D instead.)

PMEnd (page 2148) Deprecated in Mac OS X v10.4
Closes Core Printing and releases its allocated memory. (Deprecated. Use PMRelease (page 2214) to release a PMPrintSession (page 2277) object instead.)

PMError (page 2148) Deprecated in Mac OS X v10.4
Obtains the result code from the last printing function called by your application. (Deprecated. Use PMSessionError (page 2229) instead.)

PMGeneral (page 2152) Deprecated in Mac OS X v10.4
Maintains compatibility with the PrGeneral function in the classic Printing Manager. (Deprecated. Use PMPrinterGetCommInfo (page 2190) instead.)

PMGetColorMode (page 2155) Deprecated in Mac OS X v10.4
Obtains the color mode for the print job. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

PMGetDriverCreator (page 2157) Deprecated in Mac OS X v10.4
Obtains the creator of the driver associated with the current printer. (Deprecated. Use PMPrinterGetDriverCreator (page 2192) instead.)

PMGetDriverReleaseInfo (page 2157) Deprecated in Mac OS X v10.4
Obtains release information for the driver associated with the current printer. (Deprecated. Use PMPrinterGetDriverReleaseInfo (page 2193) instead.)

PMGetGrafPtr (page 2159) Deprecated in Mac OS X v10.4
Obtains the printing port from an opaque printing context. (Deprecated. Use PMSessionGetCGGraphicsContext (page 2230) instead.)

PMGetIndexedPrinterResolution (page 2160) Deprecated in Mac OS X v10.4
Obtains a resolution setting based on an index into the range of settings supported by the current printer. (Deprecated. Use PMPrinterGetIndexedPrinterResolution (page 2194) instead.)

PMGetJobName (page 2160) Deprecated in Mac OS X v10.4
Obtains the name of the print job. (Deprecated. Use PMPrintSettingsGetJobName (page 2210) instead.)

PMGetLanguageInfo (page 2162) Deprecated in Mac OS X v10.4
Obtains information about the current printer's imaging language. (Deprecated. Use PMPrinterGetLanguageInfo (page 2194) instead.)

PMGetPhysicalPageSize (page 2166) Deprecated in Mac OS X v10.4
Obtains the size of the imageable area in points, unaffected by rotation, resolution, or scaling. (Deprecated. Use PMGetUnadjustedPageRect (page 2170) or examine the paper returned by PMGetPageFormatPaper (page 2164).)

PMGetPhysicalPaperSize (page 2166) Deprecated in Mac OS X v10.4
Obtains the size of the paper in points, unaffected by rotation, resolution, or scaling. (Deprecated. Use PMGetUnadjustedPaperRect (page 2171) or examine the paper returned by PMGetPageFormatPaper (page 2164).)

PMGetPrinterResolution (page 2167) Deprecated in Mac OS X v10.4
Obtains the resolution setting for the current printer according to the tag parameter. (Deprecated. Use PMPrinterGetPrinterResolutionCount (page 2199) and PMPrinterGetIndexedPrinterResolution (page 2194) to examine the available printer resolutions.)

PMGetPrinterResolutionCount (page 2168) Deprecated in Mac OS X v10.4
Obtains the number of resolution settings supported by the current printer. (Deprecated. Use PMPrinterGetPrinterResolutionCount (page 2199) instead.)

PMIsPostScriptDriver (page 2172) Deprecated in Mac OS X v10.4
Reports whether the current printer driver supports the PostScript language. (Deprecated. Use PMPrinterIsPostScriptCapable (page 2201) or PMPrinterIsPostScriptPrinter (page 2202) instead.)

PMMakeOldPrintRecord (page 2172) Deprecated in Mac OS X v10.4
Creates an old-style print record from a PMPageFormat and a PMPrintSettings object. (Deprecated. There is no replacement; old-style print records are obsolete and serve no useful purpose in Mac OS X.)

PMNewPageFormat (page 2173) Deprecated in Mac OS X v10.4
Creates a new PMPageFormat object. (Deprecated. Use PMCreatePageFormat (page 2143) instead.)

PMNewPrintSettings (page 2173) Deprecated in Mac OS X v10.4
Creates a new PMPrintSettings object. (Deprecated. Use PMCreatePrintSettings (page 2144) instead.)

PMPostScriptBegin (page 2183) Deprecated in Mac OS X v10.4
Puts the current driver into PostScript mode, ready to accept PostScript data instead of QuickDraw data. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMPostScriptData (page 2183) Deprecated in Mac OS X v10.4
Passes PostScript data, referenced by a pointer, to the current printer driver. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMPostScriptEnd (page 2184) Deprecated in Mac OS X v10.4
Restores the current driver to QuickDraw mode, ready to accept QuickDraw data instead of PostScript data. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMPostScriptFile (page 2184) Deprecated in Mac OS X v10.4
Passes PostScript data, contained in a file, to the current printer driver. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMPostScriptHandle (page 2185) Deprecated in Mac OS X v10.4
Passes PostScript data, referenced by a handle, to the current printer driver. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSessionGetDocumentFormatSupported (page 2234) Deprecated in Mac OS X v10.4
Obtains the spool file formats that are accepted by the current printer driver. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSessionIsDocumentFormatSupported (page 2235) Deprecated in Mac OS X v10.4
Reports whether the current printer driver supports a specified spool file format. (Deprecated. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

PMSetColorMode (page 2251) Deprecated in Mac OS X v10.4
Sets the desired color mode for the print job. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

`PMSetError` (page 2253) Deprecated in Mac OS X v10.4
: Sets the value of the current result code. (Deprecated. Use `PMSessionSetError` (page 2246) instead.)

`PMSetIdleProc` (page 2255) Deprecated in Mac OS X v10.4
: Installs an idle callback function in your print loop. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

`PMSetJobName` (page 2255) Deprecated in Mac OS X v10.4
: Specifies the name of the print job. (Deprecated. Use `PMPrintSettingsSetJobName` (page 2211) instead.)

`PMSetPhysicalPaperSize` (page 2260) Deprecated in Mac OS X v10.4
: Requests a particular paper size, unaffected by rotation, resolution, or scaling. (Deprecated. Use `PMCreatePageFormatWithPMPaper` (page 2144) instead.)

`PMValidatePageFormat` (page 2269) Deprecated in Mac OS X v10.4
: Obtains a valid `PMPageFormat` object. (Deprecated. Use `PMSessionValidatePageFormat` (page 2248) instead.)

`PMValidatePrintSettings` (page 2270) Deprecated in Mac OS X v10.4
: Obtains a valid `PMPrintSettings` object. (Deprecated. Use `PMSessionValidatePrintSettings` (page 2249) instead.)

Functions

DisposePMIdleUPP

Disposes of a universal procedure pointer (UPP) to an idle callback. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
void DisposePMIdleUPP (
   PMIdleUPP userUPP
);
```

Discussion

You do not need this function in Mac OS X. Instead, use the standard idle proc. See the `PMIdleProcPtr` (page 2273) callback function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PMCoreDeprecated.h`

InvokePMIdleUPP

Calls an idle callback. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
void InvokePMIdleUPP (
   PMIdleUPP userUPP
);
```

Discussion
You do not need this function in Mac OS X. Instead, use the standard idle proc. See the `PMIdleProcPtr` (page 2273) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

NewPMIdleUPP

Creates a new universal procedure pointer (UPP) to an idle callback. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
PMIdleUPP NewPMIdleUPP (
   PMIdleProcPtr userRoutine
);
```

Discussion
You do not need this function in Mac OS X. Instead, use the standard idle proc. See the `PMIdleProcPtr` (page 2273) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMBegin

Prepares Core Printing for use. (Deprecated in Mac OS X v10.4. Use `PMCreateSession` (page 2145) instead.)

```
OSStatus PMBegin ();
```

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Do not nest calls to `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMCGImageCreateWithEPSDataProvider

Creates an image that references both the PostScript contents of EPS data and a preview (proxy) image for the data.

```
CGImageRef PMCGImageCreateWithEPSDataProvider (
   CGDataProviderRef epsDataProvider,
   CGImageRef epsPreview
);
```

Parameters

epsDataProvider

A Quartz data provider that supplies the PostScript contents of the EPS file. The EPS data must begin with the EPSF required header and bounding box DSC (Document Structuring Conventions) comments.

epsPreview

A Quartz image that serves as the proxy image for the EPS file. When the image returned by this function is rendered onscreen or sent to a printer that cannot render PostScript, this proxy image is drawn instead.

Return Value

An image capable of rendering either the EPS content or the proxy image, depending upon the capabilities of the destination printer.

Discussion

It is likely that data will not be read from the EPS data provider until after this function returns. You should be careful not to free the underlying EPS data until the data provider's release function is invoked. Similarly, do not free the preview image data until the image data provider's release function is invoked. You are responsible for releasing the data providers for the EPS image and the EPS preview image.

Note that in Mac OS X v10.3 and later, Quartz can convert EPS data into PDF data. Using this feature and then using Quartz to draw the resulting PDF data may produce superior results for your application. See *CGPSConverter Reference* for details.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`PMCore.h`

PMConvertOldPrintRecord

Creates a new `PMPageFormat` object and a new `PMPrintSettings` object from a print record created by the classic Printing Manager. (Deprecated in Mac OS X v10.4. There is no replacement; during the transition from Mac OS 9 to Mac OS X, this function facilitated the migration of print records saved in documents created in Mac OS 9, but the function no longer serves any useful purpose in Mac OS X.)

```
OSStatus PMConvertOldPrintRecord (
    Handle printRecordHandle,
    PMPrintSettings *printSettings,
    PMPageFormat *pageFormat
);
```

Parameters

printRecordHandle

A handle to a print record created by the classic Printing Manager.

printSettings
On return, a validated `PMPrintSettings` object.

pageFormat
On return, a validated `PMPageFormat` object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMCopyAvailablePPDs

Obtains the list of PostScript printer description (PPD) files in a PPD domain.

```
OSStatus PMCopyAvailablePPDs (
   PMPPDDomain domain,
   CFArrayRef *ppds
);
```

Parameters

domain
The PPD domain to search. See "PostScript Printer Description File Domains" (page 2292) for a description of the constants you can use to specify the domain.

ppds
A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array of PPD files in the specified domain. Each element in the array is a Core Foundation URL object that specifies the location of a PPD file or a compressed PPD file. You are responsible for releasing the array. If the specified domain is not valid, the variable is set to `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMCopyLocalizedPPD

Obtains a localized PostScript printer description (PPD) file.

```
OSStatus PMCopyLocalizedPPD (
   CFURLRef ppd,
   CFURLRef *localizedPPD
);
```

Parameters

ppd

A Core Foundation URL object for a PPD file. You can obtain a PPD URL using the function `PMCopyAvailablePPDs` (page 2140).

localizedPPD

A pointer to your `CFURLRef` variable. On return, the variable refers to a Core Foundation URL object. The URL specifies the location of a PPD file or a compressed PPD file that has been localized for the current user's language preference. You are responsible for releasing the URL. If the *ppd* parameter is not valid, the variable is set to `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
To access the data in the PPD file, you can use the function `PMCopyPPDData` (page 2142).

Special Considerations

In Mac OS X v10.5 and later, the printing system supports globalized PPD files as defined in CUPS version 1.2 and later. A globalized PPD file contains multiple localizations within a single file. If a globalized PPD file exists, this function returns the URL to this file and it is up to the application to obtain the correct localized data. For more information, see CUPS PPD Extensions.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMCopyPageFormat

Copies the settings from one page format object into another.

```
OSStatus PMCopyPageFormat (
   PMPageFormat formatSrc,
   PMPageFormat formatDest
);
```

Parameters

formatSrc

The page format object to duplicate.

formatDest

The page format object to receive the copied settings. On return, this object contains the same settings as the `formatSrc` object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.

Declared In
PMCore.h

PMCopyPPDData

Obtains the uncompressed PPD data for a PostScript printer description (PPD) file.

```
OSStatus PMCopyPPDData (
   CFURLRef ppd,
   CFDataRef *data
);
```

Parameters

ppd

A URL for a PPD or compressed PPD file. You can obtain a PPD URL using the function PMCopyAvailablePPDs (page 2140) or PMCopyLocalizedPPD (page 2140).

data

A pointer to your CFDataRef variable. On return, the variable refers to a Core Foundation data object containing the uncompressed PPD data from the specified PPD file. You are responsible for releasing the data object. If the *ppd* parameter does not reference a PPD file, the variable is set to NULL.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
PMCore.h

PMCopyPrintSettings

Copies the settings from one print settings object into another.

```
OSStatus PMCopyPrintSettings (
   PMPrintSettings settingSrc,
   PMPrintSettings settingDest
);
```

Parameters

settingSrc

The print settings object to duplicate.

settingDest

The print settings object to receive the copied settings. On return, this object contains the same settings as the settingSrc object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.

Declared In
PMCore.h

PMCreateGenericPrinter

Creates a generic printer object.

```
OSStatus PMCreateGenericPrinter (
    PMPrinter *printer
);
```

Parameters

printer

A pointer to your PMPrinter (page 2276) variable. On return, the variable refers to a new printer object that represents the generic formatting printer. You are responsible for releasing the printer object with the function PMRelease (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function provides a way to create a PMPrinter object that represents the generic formatting printer.

Availability
Available in Mac OS X v10.5 and later.

Declared In
PMCore.h

PMCreatePageFormat

Creates a new page format object.

```
OSStatus PMCreatePageFormat (
   PMPageFormat *pageFormat
);
```

Parameters

pageFormat

A pointer to your PMPageFormat (page 2275) variable. On return, the variable refers to a new page format object. You are responsible for releasing the page format object with the function PMRelease (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function allocates memory for a new page format object in your application's memory space and sets its reference count to 1. The new page format object is empty and unusable until you call PMSessionDefaultPageFormat (page 2225) or PMCopyPageFormat (page 2141).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
PMCore.h

PMCreatePageFormatWithPMPaper

Creates a page format object with a specified paper.

```
OSStatus PMCreatePageFormatWithPMPaper (
   PMPageFormat *pageFormat,
   PMPaper paper
);
```

Parameters

pageFormat

A pointer to your PMPageFormat (page 2275) variable. On return, the variable refers to a new page format object that represents the specified paper. You are responsible for releasing the page format object with the function PMRelease (page 2214).

paper

The type of paper for the new page format object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
PMCore.h

PMCreatePrintSettings

Creates a new print settings object.

```
OSStatus PMCreatePrintSettings (
   PMPrintSettings *printSettings
);
```

Parameters

printSettings

A pointer to your PMPrintSettings (page 2277) variable. On return, the variable refers to a new print settings object. You are responsible for releasing the print settings object with the function PMRelease (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function allocates memory for a new print settings object in your application's memory space and sets its reference count to 1. The new print settings object is empty and unusable until you call PMSessionDefaultPrintSettings (page 2225) or PMCopyPrintSettings (page 2142).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMCreateSession

Creates and initializes a printing session object and creates a context for printing operations.

```
OSStatus PMCreateSession (
   PMPrintSession *printSession
);
```

Parameters

printSession

A pointer to your `PMPrintSession` (page 2277) variable. On return, the variable refers to a new printing session object. You are responsible for releasing the printing session object with the function `PMRelease` (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function allocates memory for a new printing session object in your application's memory space and sets its reference count to 1. The new printing session object is initialized with information that the printing system uses for a print job.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMDefaultPageFormat

Assigns default parameter values to an existing `PMPageFormat` object, for the current printer. (Deprecated in Mac OS X v10.4. Use `PMSessionDefaultPageFormat` (page 2225) instead.)

```
OSStatus PMDefaultPageFormat (
    PMPageFormat pageFormat
);
```

Parameters

pageFormat

On return, a `PMPageFormat` object containing default parameter values.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin` and creating a page format object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMDefaultPrintSettings

Assigns default parameter values to a `PMPrintSettings` object. (Deprecated in Mac OS X v10.4. Use `PMSessionDefaultPrintSettings` (page 2225) instead.)

```
OSStatus PMDefaultPrintSettings (
    PMPrintSettings printSettings
);
```

Parameters
printSettings
A `PMPrintSettings` object. On return, the object contains default parameter values.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin` and creating a print settings object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMDisableColorSync

Disables ColorSync color matching for the current page. (Deprecated in Mac OS X v10.4. There is no replacement; draw using Quartz 2D instead.)

```
OSStatus PMDisableColorSync ();
```

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid between calls to `PMBeginPage` and `PMEndPage`.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMDisposePageFormat

Releases memory previously allocated for a `PMPageFormat` object. (Deprecated in Mac OS X v10.4. Use `PMRelease` (page 2214) instead.)

```
OSStatus PMDisposePageFormat (
    PMPageFormat pageFormat
);
```

Parameters

pageFormat

On return, an invalidated `PMPageFormat` object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin` and creating a page format object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMDisposePrintSettings

Releases memory previously allocated for a `PMPrintSettings` object. (Deprecated in Mac OS X v10.4. Use `PMRelease` (page 2214) instead.)

```
OSStatus PMDisposePrintSettings (
    PMPrintSettings printSettings
);
```

Parameters

printSettings

On return, an invalidated `PMPrintSettings` reference.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin` and creating a print settings object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMEnableColorSync

Enables ColorSync color matching for the current page. (Deprecated in Mac OS X v10.4. There is no replacement; draw using Quartz 2D instead.)

```
OSStatus PMEnableColorSync ();
```

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid between calls to `PMBeginPage` and `PMEndPage`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMEnd

Closes Core Printing and releases its allocated memory. (Deprecated in Mac OS X v10.4. Use `PMRelease` (page 2214) to release a `PMPrintSession` (page 2277) object instead.)

```
OSStatus PMEnd (void);
```

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMError

Obtains the result code from the last printing function called by your application. (Deprecated in Mac OS X v10.4. Use `PMSessionError` (page 2229) instead.)

```
OSStatus PMError ();
```

Return Value
A result code. The result code `kPMCancel` indicates the user canceled the current print job.

Discussion
Valid after calling `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMFlattenPageFormat

Flattens a page format object into a Memory Manager handle for storage in a user document. (Deprecated in Mac OS X v10.5. Use `PMPageFormatCreateDataRepresentation` (page 2174) instead.)

```
OSStatus PMFlattenPageFormat (
   PMPageFormat pageFormat,
   Handle *flatFormat
);
```

Parameters

pageFormat

The page format object to flatten.

flatFormat

A pointer to your `Handle` variable. On return, the variable refers to a Memory Manager handle that contains the flattened page format object. The handle is allocated by the function. You are responsible for disposing of the handle.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`PMCoreDeprecated.h`

PMFlattenPageFormatToCFData

Flattens a page format object into a Core Foundation data object for storage in a user document. (Deprecated in Mac OS X v10.5. Use `PMPageFormatCreateDataRepresentation` (page 2174) instead.)

```
OSStatus PMFlattenPageFormatToCFData (
   PMPageFormat pageFormat,
   CFDataRef *flatFormat
);
```

Parameters

pageFormat

The page format object to flatten.

flatFormat

A pointer to your `CFDataRef` variable. On return, the variable refers to a Core Foundation data object containing a flattened representation of the specified page format object. You are responsible for releasing the data object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.5.

See Also
`PMUnflattenPageFormatWithCFData` (page 2266)

Declared In
`PMCoreDeprecated.h`

PMFlattenPageFormatToURL

Flattens a page format object into a file for storage in a user document. (Deprecated in Mac OS X v10.5. Use `PMPageFormatCreateDataRepresentation` (page 2174) and write the resulting data to your destination.)

```
OSStatus PMFlattenPageFormatToURL (
   PMPageFormat pageFormat,
   CFURLRef flattenFileURL
);
```

Parameters

pageFormat

The page format object to flatten.

flatFormat

A Core Foundation URL specifying a file to contain a flattened representation of the specified page format object. If the file already exists, it is overwritten. Only file-based URLs are supported.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.5.

See Also
`PMUnflattenPageFormatWithURL` (page 2266)

Declared In
`PMCoreDeprecated.h`

PMFlattenPrintSettings

Flattens a print settings object into a Memory Manager handle for storage in a user document. (Deprecated in Mac OS X v10.5. Use `PMPrintSettingsCreateDataRepresentation` (page 2208) instead.)

```
OSStatus PMFlattenPrintSettings (
   PMPrintSettings printSettings,
   Handle *flatSettings
);
```

Parameters

printSettings

The print settings object to flatten.

flatSettings

A pointer to your `Handle` variable. On return, the variable refers to a Memory Manager handle that contains a flattened print settings object. The handle is allocated by the function. You are responsible for disposing of the handle.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

There are no scoping requirements as to when you may use this function.

Apple recommends that you do not reuse the print settings information if the user prints the document again. The information supplied by the user in the Print dialog should pertain to the document only while the document prints, so there is no need to save the print settings object.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMFlattenPrintSettingsToCFData

Flattens a print settings object into a Core Foundation data object for storage in a user document. (Deprecated in Mac OS X v10.5. Use `PMPrintSettingsCreateDataRepresentation` (page 2208) instead.)

```
OSStatus PMFlattenPrintSettingsToCFData (
   PMPrintSettings printSettings,
   CFDataRef *flatSetting
);
```

Parameters

printSettings

The print settings object to flatten.

flatSetting

A pointer to your `CFDataRef` variable. On return, the variable refers to a Core Foundation data object that contains a flattened representation of the specified print settings object. You are responsible for releasing the data object.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.4 and later.

Deprecated in Mac OS X v10.5.

See Also
PMUnflattenPrintSettingsWithCFData (page 2268)

Declared In
PMCoreDeprecated.h

PMFlattenPrintSettingsToURL

Flattens a print settings object into a URL for storage in a user document. (Deprecated in Mac OS X v10.5. Instead use PMPrintSettingsCreateDataRepresentation (page 2208) and write the resulting data to your destination.)

```
OSStatus PMFlattenPrintSettingsToURL (
   PMPrintSettings printSettings,
   CFURLRef flattenFileURL
);
```

Parameters

printSettings

The print settings object to flatten.

flattenFileURL

A Core Foundation URL specifying a file to contain a flattened representation of the specified print settings object. If the file already exists, it is overwritten. Only file-based URLs are supported.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.5.

See Also
PMUnflattenPrintSettingsWithURL (page 2268)

Declared In
PMCoreDeprecated.h

PMGeneral

Maintains compatibility with the PrGeneral function in the classic Printing Manager. (Deprecated in Mac OS X v10.4. Use PMPrinterGetCommInfo (page 2190) instead.)

```
OSStatus PMGeneral (
    Ptr pData
);
```

Parameters

pData

A pointer to a PrGeneral data structure.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMGetAdjustedPageRect

Obtains the imageable area or page rectangle, taking into account orientation, application drawing resolution, and scaling settings.

```
OSStatus PMGetAdjustedPageRect (
   PMPageFormat pageFormat,
   PMRect *pageRect
);
```

Parameters

pageFormat
The page format object whose adjusted page rectangle you want to obtain.

pageRect
A pointer to your `PMRect` (page 2277) structure. On return, the structure contains the current imageable area, in points, taking into account scaling, rotation, and application resolution settings. The page rectangle is the area of the page to which an application can draw. The coordinates for the upper-left corner of the page rectangle are (0,0). See *Supporting Printing in Your Carbon Application* for more information on page and paper rectangles.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Before using this function, you must call `PMSessionValidatePageFormat` (page 2248) to ensure that the values for the adjusted page rectangle correctly account for scaling, rotation, and application resolution settings.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMGetAdjustedPaperRect

Obtains the rectangle defining the paper size, taking into account orientation, application drawing resolution, and scaling settings.

```
OSStatus PMGetAdjustedPaperRect (
   PMPageFormat pageFormat,
   PMRect *paperRect
);
```

Parameters

pageFormat

The page format object whose adjusted paper rectangle you want to obtain.

paperRect

A pointer to your `PMRect` (page 2277) structure. On return, the structure describes the current paper size, in points, taking into account scaling, rotation, and application resolution settings. The coordinates of the upper-left corner of the paper rectangle are specified relative to the page rectangle. The coordinates of the upper-left corner of the page rectangle are always (0,0), which means the coordinates of the upper-left corner of the paper rectangle are always negative or (0,0). See *Supporting Printing in Your Carbon Application* for more information on page and paper rectangles.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Before using this function, you must call the function `PMSessionValidatePageFormat` (page 2248) to ensure that the values for the adjusted paper rectangle correctly account for scaling, rotation, and application resolution settings.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PMCore.h`

PMGetCollate

Obtains a Boolean value that indicates whether the job collate option is selected.

```
OSStatus PMGetCollate (
   PMPrintSettings printSettings,
   Boolean *collate
);
```

Parameters

printSettings

The print settings object you're querying to determine whether the job collate option is selected.

collate

A pointer to your Boolean variable. On return, `true` if the job collate option is selected; otherwise, `false`.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The Collated checkbox is displayed in the Copies & Pages pane of the Print dialog. This option determines how printed material is organized. For example, if you have a document that is three pages long and you are printing multiple copies with the Collated option selected, the job prints pages 1, 2, and 3 in that order and then repeats. However, if the Collated option is not selected and you're printing multiple copies of those same three pages, the job prints copies of page 1, then copies of page 2, and finally copies of page 3.

Availability
Available in Mac OS X v10.2 and later.

See Also
`PMSetCollate` (page 2251)

Declared In
`PMCore.h`

PMGetColorMode

Obtains the color mode for the print job. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
OSStatus PMGetColorMode (
   PMPrintSettings printSettings,
   PMColorMode *colorMode
);
```

Parameters

printSettings

The print settings object whose color mode you want to obtain.

colorMode

On return, a pointer to a value that represents the color mode setting. See "Color Modes" (page 2298) for a list of possible return values.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is not recommended. It doesn't do anything in Mac OS X and in general is no longer appropriate for applications to call.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMGetCopies

Obtains the number of copies that the user requests to be printed.

```
OSStatus PMGetCopies (
   PMPrintSettings printSettings,
   UInt32 *copies
);
```

Parameters

printSettings

The print settings object whose number of copies you want to obtain.

copies

A pointer to your `UInt32` variable. On return, the variable contains the number of copies requested by the user.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.0 and later.

See Also

`PMSetCopies` (page 2252)

Declared In

`PMCore.h`

PMGetDestination

Obtains the output destination of a print job. (Deprecated in Mac OS X v10.5. Use `PMSessionGetDestinationType` (page 2232), `PMSessionCopyDestinationFormat` (page 2221), or `PMSessionCopyDestinationLocation` (page 2221) instead.)

```
OSStatus PMGetDestination (
   PMPrintSettings printSettings,
   PMDestinationType *destType,
   CFURLRef *fileURL
);
```

Parameters

printSettings

The print settings object whose destination you want to obtain.

destType

A pointer to your `PMDestinationType` variable. On return, the variable indicates the destination for the print job. See "Destination Types" (page 2281).

fileURL

A pointer to your `CFURLRef` variable. On return, the variable refers to a Core Foundation URL that contains the location of the print job destination.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

For print jobs that are sent to disk, as opposed a printer, you can use this function to obtain the location of the destination file. Valid within a printing session after creating a print settings object.

Before using this function you must call `PMSessionValidatePrintSettings` or `PMValidatePrintSettings` to ensure that the print settings object is valid.

Special Considerations

This function does not take a print session parameter and therefore cannot indicate whether preview has been selected as the destination.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMGetDriverCreator

Obtains the creator of the driver associated with the current printer. (Deprecated in Mac OS X v10.4. Use `PMPrinterGetDriverCreator` (page 2192) instead.)

```
OSStatus PMGetDriverCreator (
    OSType *creator
);
```

Parameters

creator
On return, the 4-byte creator type of the driver (for example, `'APPL'` for an Apple printer driver).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMGetDriverReleaseInfo

Obtains release information for the driver associated with the current printer. (Deprecated in Mac OS X v10.4. Use `PMPrinterGetDriverReleaseInfo` (page 2193) instead.)

```
OSStatus PMGetDriverReleaseInfo (
    VersRec *release
);
```

Parameters

release

On return, a pointer to a `VersRec` data structure containing the driver's short and long version strings and country code.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMGetDuplex

Obtains the selected duplex mode.

```
OSStatus PMGetDuplex (
   PMPrintSettings printSettings,
   PMDuplexMode *duplexSetting
);
```

Parameters

printSettings

The print settings object whose duplex mode you want to obtain.

duplexSetting

A pointer to your `PMDuplexMode` variable. On return, the variable contains the duplex mode setting in the current print job. Possible values include:

- `kPMDuplexNone` (one-sided printing)
- `kPMDuplexNoTumble` (two-sided printing)
- `kPMDuplexTumble` (two-sided printing with tumbling)

See "Duplex Modes" (page 2282) for a full description of the duplex mode constants.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Duplex printing is a print job that prints on both sides of the paper. The Two-Sided printing control is displayed in the Layout pane of the Print dialog.

Availability
Available in Mac OS X v10.4 and later.

Declared In
PMCore.h

PMGetFirstPage

Obtains the number of the first page to be printed.

```
OSStatus PMGetFirstPage (
   PMPrintSettings printSettings,
   UInt32 *first
);
```

Parameters

printSettings

The print settings object whose first page number you want to obtain.

first

A pointer to your UInt32 variable. On return, the variable contains the page number of the first page to print. The default first page number is 1.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can use this function to obtain the page number entered by the user in the From field of the Print dialog. If the user selects the All button, the function returns a value of 1. If the user did not enter a value, the function returns the value of the previous call to PMSetFirstPage, if any, or the default value of 1.

Availability
Available in Mac OS X v10.0 and later.

See Also
PMSetFirstPage (page 2254)

Related Sample Code
CarbonSketch

Declared In
PMCore.h

PMGetGrafPtr

Obtains the printing port from an opaque printing context. (Deprecated in Mac OS X v10.4. Use PMSessionGetCGGraphicsContext (page 2230) instead.)

```
OSStatus PMGetGrafPtr (
    PMPrintContext printContext,
    GrafPtr *grafPort
);
```

Parameters

printContext

A PMPrintContext object.

grafPort
On return, a pointer to a `grafPort` defining the current printing port.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin` and creating a printing context.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMGetIndexedPrinterResolution

Obtains a resolution setting based on an index into the range of settings supported by the current printer. (Deprecated in Mac OS X v10.4. Use `PMPrinterGetIndexedPrinterResolution` (page 2194) instead.)

```
OSStatus PMGetIndexedPrinterResolution (
    UInt32 index,
    PMResolution *res
);
```

Parameters
index
An index into the range of resolution settings supported by the specified printer. Index values begin at 1.

res
On return, the printer resolution setting.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`. You must first use the `PMGetPrinterResolutionCount` function to obtain the number of resolution settings supported by the current printer.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMGetJobName

Obtains the name of the print job. (Deprecated in Mac OS X v10.4. Use `PMPrintSettingsGetJobName` (page 2210) instead.)

```
OSStatus PMGetJobName (
   PMPrintSettings printSettings,
   StringPtr name
);
```

Parameters

printSettings

A `PMPrintSettings` object.

name

On return, the name of the print job.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Valid after calling `PMBegin` and creating a print settings object. Before using this function you must call `PMValidatePrintSettings` to ensure that the print settings object is valid.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMGetJobNameCFString

Obtains the name of the print job. (Deprecated in Mac OS X v10.5. Use `PMPrintSettingsGetJobName` (page 2210) instead.)

```
OSStatus PMGetJobNameCFString (
   PMPrintSettings printSettings,
   CFStringRef *name
);
```

Parameters

printSettings

The print settings object whose job name you want to obtain.

name

A pointer to your `CFStringRef` variable. On return, the variable refers to a string that contains the name of the print job. Despite what its name implies, the function `PMGetJobNameCFString` has Create/Copy semantics which means your application must release the string returned to it.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

See Also

`PMSetJobNameCFString` (page 2256)

Declared In
PMCoreDeprecated.h

PMGetLanguageInfo

Obtains information about the current printer's imaging language. (Deprecated in Mac OS X v10.4. Use PMPrinterGetLanguageInfo (page 2194) instead.)

```
OSStatus PMGetLanguageInfo (
    PMLanguageInfo *info
);
```

Parameters

info

On return, a pointer to a data structure containing the printer's language level, version and release. The format of the returned data is based on the PostScript language.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling PMBegin. PMGetLanguageInfo is useful for PostScript printers but may be irrelevant for other types of printers.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMGetLastPage

Obtains the number of the last page to be printed.

```
OSStatus PMGetLastPage (
   PMPrintSettings printSettings,
   UInt32 *last
);
```

Parameters

printSettings

The print settings object whose last page number you want to obtain.

last

A pointer to your UInt32 variable. On return, the variable contains the page number of the last page to print.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You use this function to obtain the page number entered by the user in the To field of the Print dialog. If the user did not enter a value, the function returns the value of the previous call to PMSetLastPage, if any, or a default value.

You should not look for the constant `kPMPrintAllPages`. That constant is used only with the `PMSetLastPage` and `PMSetPageRange` functions to specify a last page. It is not returned by the `PMGetLastPage` function.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMSetLastPage` (page 2257)

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMGetOrientation

Obtains the current setting for page orientation.

```
OSStatus PMGetOrientation (
   PMPageFormat pageFormat,
   PMOrientation *orientation
);
```

Parameters

`pageFormat`

The page format object whose orientation you want to obtain.

`orientation`

A pointer to your `PMOrientation` variable. On return, the variable contains a constant value indicating the page orientation. Supported values are:

- `kPMPortrait`
- `kPMLandscape`
- `kPMReversePortrait` (supported in Mac OS X v10.5 and later)
- `kPMReverseLandscape`

See "Page Orientation Constants" (page 2283) for a complete description of the page orientation constants.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMSetOrientation` (page 2257)

Declared In
`PMCore.h`

PMGetPageFormatExtendedData

Obtains extended page format data previously stored by your application.

```
OSStatus PMGetPageFormatExtendedData (
   PMPageFormat pageFormat,
   OSType dataID,
   UInt32 *size,
   void *extendedData
);
```

Parameters

pageFormat

The page format object that contains your extended data.

dataID

A 4-character code that identifies your data. This is typically your application's creator code. If your creator code is outside the ASCII 7-bit character range 0x20–0x7F, you need to use a different 4-character code.

size

A pointer to a value that specifies the size of the buffer you have allocated for the extended page format data. On return, this variable contains the number of bytes read into the buffer or the size of the extended data. You can pass the constant `kPMDontWantSize` if you do not need this information. (See "Data Not Wanted Constants" (page 2279) for more information.)

extendedData

A pointer to a buffer to receive the extended data. Pass the constant `kPMDontWantData` if you do not want to read the data. (See "Data Not Wanted Constants" (page 2279) for more information.)

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Your application typically needs to call the function `PMGetPageFormatExtendedData` two times in order to retrieve the extended page format data. The first time, pass the constant `kPMDontWantData` in the parameter *extendedData* to obtain the buffer size required for the extended data. Then allocate the buffer and call the function a second time to read the extended data into your buffer.

If you write a printing dialog extension for your application that stores data in the page format object, you use the function `PMGetPageFormatExtendedData` to retrieve the data associated with it.

Availability

Available in Mac OS X v10.0 and later.

See Also

`PMSetPageFormatExtendedData` (page 2258)

Declared In

`PMCore.h`

PMGetPageFormatPaper

Obtains the paper associated with a page format object.

```
OSStatus PMGetPageFormatPaper (
   PMPageFormat format,
   PMPaper *paper
);
```

Parameters

pageFormat

The page format object whose paper you want to obtain.

paper

A pointer to your `PMPaper` (page 2275) variable. On return, the variable refers to a paper object that represents the paper associated with the specified page format. You should not release the paper object without first retaining it.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`PMCore.h`

PMGetPageRange

Obtains the valid range of pages that can be printed.

```
OSStatus PMGetPageRange (
   PMPrintSettings printSettings,
   UInt32 *minPage,
   UInt32 *maxPage
);
```

Parameters

printSettings

The print settings object whose page range you want to obtain.

minPage

A pointer to your `UInt32` variable. On return, the variable contains the minimum page number allowed.

maxPage

A pointer to your `UInt32` variable. On return, the variable contains the maximum page number allowed.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

The page range returned by the function `PMGetPageRange` is independent of the first and last page values returned by `PMGetFirstPage` (page 2159) and `PMGetLastPage` (page 2162). See `PMSetPageRange` for more information.

Availability

Available in Mac OS X v10.0 and later.

See Also
PMSetPageRange (page 2259)

Declared In
PMCore.h

PMGetPhysicalPageSize

Obtains the size of the imageable area in points, unaffected by rotation, resolution, or scaling. (Deprecated in Mac OS X v10.4. Use PMGetUnadjustedPageRect (page 2170) or examine the paper returned by PMGetPageFormatPaper (page 2164).)

```
OSStatus PMGetPhysicalPageSize (
   PMPageFormat pageFormat,
   PMRect *pageSize
);
```

Parameters

pageFormat

A PMPageFormat object previously created by your application.

pageSize

On return, a rectangle describing the physical page size where your application can draw.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
PMCoreDeprecated.h

PMGetPhysicalPaperSize

Obtains the size of the paper in points, unaffected by rotation, resolution, or scaling. (Deprecated in Mac OS X v10.4. Use PMGetUnadjustedPaperRect (page 2171) or examine the paper returned by PMGetPageFormatPaper (page 2164).)

```
OSStatus PMGetPhysicalPaperSize (
   PMPageFormat pageFormat,
   PMRect *paperSize
);
```

Parameters

pageFormat

A PMPageFormat object previously created by your application.

paperSize

On return, a rectangle describing the physical size of the paper. Units are in 1/72 inch. Thus a 8.5 x 11 sheet of paper returns for its individual components:

top - 0.0

left - 0.0

bottom - 792.0

right - 612.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMGetPrinterResolution

Obtains the resolution setting for the current printer according to the tag parameter. (Deprecated in Mac OS X v10.4. Use `PMPrinterGetPrinterResolutionCount` (page 2199) and `PMPrinterGetIndexedPrinterResolution` (page 2194) to examine the available printer resolutions.)

```
OSStatus PMGetPrinterResolution (
    PMTag tag,
    PMResolution *res
);
```

Parameters

tag

Specifies the kind of resolution information required.

res

The printer resolution setting.

Return Value
A result code. . The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Discussion
Valid after calling `PMBegin`.

The following resolution tag constants are recognized:

`kPMMinRange`

The minimum resolution supported by the printer.

`kPMMaxRange`

The maximum resolution supported by the printer.

`kPMMinSquareResolution`

The minimum resolution setting for which the horizontal and vertical resolutions are equal.

`kPMMaxSquareResolution`
The maximum resolution setting for which the horizontal and vertical resolutions are equal.

`kPMDefaultResolution`
The default resolution setting for the printer (typically 72 dpi).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMGetPrinterResolutionCount

Obtains the number of resolution settings supported by the current printer. (Deprecated in Mac OS X v10.4. Use `PMPrinterGetPrinterResolutionCount` (page 2199) instead.)

```
OSStatus PMGetPrinterResolutionCount (
    UInt32 *count
);
```

Parameters

count
On return, the number of supported printing resolutions.

Return Value
A result code. The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Discussion
Valid after calling `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMGetPrintSettingsExtendedData

Obtains extended print settings data previously stored by your application.

```
OSStatus PMGetPrintSettingsExtendedData (
   PMPrintSettings printSettings,
   OSType dataID,
   UInt32 *size,
   void *extendedData
);
```

Parameters

printSettings
The print settings object whose extended data you want to obtain.

dataID

The unique 4-character code of the data to retrieve. This is typically your application's creator code. However, if your creator code is outside the ASCII 7-bit character range 0x20–0x7F, you need to use a different 4-character code.

size

A pointer to a value that specifies the size of the buffer you have allocated for the extended print settings data. On return, this variable contains the number of bytes read into the buffer or the size of the extended data. You can pass the constant `kPMDontWantSize` if you do not need this information. (See "Data Not Wanted Constants" (page 2279) for more information.)

extendedData

A pointer to a buffer to receive the extended data. Pass the constant `kPMDontWantData` if you do not want to read the data. (See "Data Not Wanted Constants" (page 2279) for more information.)

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Your application typically needs to call `PMGetPrintSettingsExtendedData` two times in order to retrieve the extended print settings data. The first time, pass the constant `kPMDontWantData` in the *extendedData* parameter to obtain the buffer size required for the extended data. Then allocate the buffer and call the function a second time to read the extended data into your buffer.

You may find it easier to use the functions `PMPrintSettingsSetValue` (page 2212) and `PMPrintSettingsGetValue` (page 2210) to store and retrieve user-defined data in a print settings object. If you use these functions, make sure that the custom keys you define for your private data do not conflict with other print settings keys.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMSetPrintSettingsExtendedData` (page 2261)

Declared In
`PMCore.h`

PMGetResolution

Obtains the current application's drawing resolution. (Deprecated in Mac OS X v10.5. Draw using Quartz 2D and call `CGContextScaleCTM` (page 105) instead.)

```
OSStatus PMGetResolution (
   PMPageFormat pageFormat,
   PMResolution *res
);
```

Parameters

pageFormat

The page format object whose drawing resolution you want to obtain.

res

A pointer to your `PMResolution` (page 2278) structure. On return, the structure contains the drawing resolution of the current application.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function obtains the drawing resolution specified in the page format, not the resolution of the current printer. You can use `PMPrinterGetPrinterResolutionCount` (page 2199) and `PMPrinterGetIndexedPrinterResolution` (page 2194) to examine the available printer resolutions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMGetScale

Obtains the scaling factor currently applied to the page and paper rectangles.

```
OSStatus PMGetScale (
   PMPageFormat pageFormat,
   double *scale
);
```

Parameters

pageFormat
The page format object whose scaling factor you want to obtain.

scale
A pointer to your double-precision variable. On return, the variable contains the scaling factor expressed as a percentage. For example, a value of 100.0 means 100 percent (that is, no scaling); a value of 50.0 means 50 percent scaling.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMSetScale` (page 2263)

Declared In
`PMCore.h`

PMGetUnadjustedPageRect

Obtains the imageable area or page rectangle, unaffected by orientation, resolution, or scaling.

```
OSStatus PMGetUnadjustedPageRect (
   PMPageFormat pageFormat,
   PMRect *pageRect
);
```

Parameters

pageFormat

The page format object whose unadjusted page rectangle you want to obtain.

pageRect

A pointer to your `PMRect` (page 2277) data structure. On return, the structure contains the size of the page rectangle, in points. The page rectangle is the area of the page to which an application can draw. The coordinates for the upper-left corner of the page rectangle are (0,0). See *Supporting Printing in Your Carbon Application* for more information on page and paper rectangles.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PMCore.h`

PMGetUnadjustedPaperRect

Obtains the paper rectangle, unaffected by rotation, resolution, or scaling.

```
OSStatus PMGetUnadjustedPaperRect (
   PMPageFormat pageFormat,
   PMRect *paperRect
);
```

Parameters

pageFormat

The page format object whose unadjusted paper rectangle you want to obtain.

paperRect

A pointer to your `PMRect` (page 2277) data structure. On return, the structure contains the physical size of the paper, in points. The coordinates of the upper-left corner of the paper rectangle are specified relative to the page rectangle. The coordinates of the upper-left corner of the page rectangle are always (0,0), which means the coordinates of the upper-left corner of the paper rectangle are always negative or (0,0). See *Supporting Printing in Your Carbon Application* for more information on page and paper rectangles.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PMCore.h`

PMIsPostScriptDriver

Reports whether the current printer driver supports the PostScript language. (Deprecated in Mac OS X v10.4. Use `PMPrinterIsPostScriptCapable` (page 2201) or `PMPrinterIsPostScriptPrinter` (page 2202) instead.)

```
OSStatus PMIsPostScriptDriver (
    Boolean *isPostScript
);
```

Parameters

isPostScript
Returns `true` if the current printer driver supports PostScript.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`. In Mac OS X, `PMIsPostScriptDriver` always returns `false`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMMakeOldPrintRecord

Creates an old-style print record from a `PMPageFormat` and a `PMPrintSettings` object. (Deprecated in Mac OS X v10.4. There is no replacement; old-style print records are obsolete and serve no useful purpose in Mac OS X.)

```
OSStatus PMMakeOldPrintRecord (
    PMPrintSettings printSettings,
    PMPageFormat pageFormat,
    Handle *printRecordHandle
);
```

Parameters

printSettings
A `PMPrintSettings` object.

pageFormat
A `PMPageFormat` object.

printRecordHandle
On return, a handle to a an old-style print record. Your application must dispose of this handle.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin` and creating a page format and print settings object. Use `PMMakeOldPrintRecord` to create a print record to store with your documents for compatibility with pre-Carbon versions of your application. Note that because the page format and print settings objects contain more information than the old-style print record, some settings may be lost in conversion.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMNewPageFormat

Creates a new `PMPageFormat` object. (Deprecated in Mac OS X v10.4. Use `PMCreatePageFormat` (page 2143) instead.)

```
OSStatus PMNewPageFormat (
    PMPageFormat *pageFormat
);
```

Parameters

pageFormat
On return, an initialized `PMPageFormat` object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`. The function `PMNewPageFormat` allocates memory for a new `PMPageFormat` object in your application's memory space. The new page format object is empty until you set its values, or until you call `PMDefaultPageFormat` or `PMValidatePageFormat`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMNewPrintSettings

Creates a new `PMPrintSettings` object. (Deprecated in Mac OS X v10.4. Use `PMCreatePrintSettings` (page 2144) instead.)

```
OSStatus PMNewPrintSettings (
    PMPrintSettings *printSettings
);
```

Parameters

printSettings
On return, an initialized `PMPrintSettings` object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`. The function `PMNewPrintSettings` allocates memory for a new `PMPrintSettings` object in your application's memory space. The new print settings object is empty until you set its values, or until you call `PMDefaultPrintSettings` or `PMValidatePrintSettings`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMPageFormatCreateDataRepresentation

Creates a data representation of a page format object.

```
OSStatus PMPageFormatCreateDataRepresentation (
    PMPageFormat pageFormat,
    CFDataRef *data,
    PMDataFormat format
);
```

Parameters

pageFormat

The page format object to convert.

data

A pointer to your `CFDataRef` variable. On return, the variable refers to a new Core Foundation data object that contains a representation of the specified page format object in the specified data format. You are responsible for releasing the data object.

format

A constant that specifies the format of the data representation. Supported values are:

- `kPMDataFormatXMLDefault` (compatible with all Mac OS X versions)
- `kPMDataFormatXMLMinimal` (approximately 3-5 times smaller; compatible with Mac OS X v10.5 and later)
- `kPMDataFormatXMLCompressed` (approximately 20 times smaller; compatible with Mac OS X v10.5 and later)

See "Data Representation Formats" (page 2280) for a full description of these formats.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is typically used to convert a page format object into a data representation suitable for storage in a user document. For information about using a Core Foundation data object, see *CFData Reference*.

Before calling this function, you should call the function `PMSessionValidatePageFormat` (page 2248) to make sure the page format object contains valid values.

Availability
Available in Mac OS X v10.5 and later.

See Also
PMPageFormatCreateWithDataRepresentation (page 2175)

Declared In
PMCore.h

PMPageFormatCreateWithDataRepresentation

Creates a page format object from a data representation.

```
OSStatus PMPageFormatCreateWithDataRepresentation (
    CFDataRef data,
    PMPageFormat *pageFormat
);
```

Parameters

data

The data representation of a page format object. The data representation must have been previously created with the function PMPageFormatCreateDataRepresentation (page 2174).

pageFormat

A pointer to your PMPageFormat (page 2275) variable. On return, the variable refers to a new page format object that contains the information stored in the specified data object. You are responsible for releasing the page format object with the function PMRelease (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is typically used to convert a data representation stored in a user document back into a page format object. For information about creating a Core Foundation data object from raw data, see *CFData Reference*.

After calling this function, you should call the function PMSessionValidatePageFormat (page 2248) to make sure the page format object contains valid values.

Availability
Available in Mac OS X v10.5 and later.

See Also
PMPageFormatCreateDataRepresentation (page 2174)

Declared In
PMCore.h

PMPageFormatGetPrinterID

Obtains the identifier of the formatting printer for a page format object.

```
OSStatus PMPageFormatGetPrinterID (
    PMPageFormat pageFormat,
    CFStringRef *printerID
);
```

Parameters

pageFormat

The page format object whose printer identifier you want to obtain.

printerID

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string that contains the identifier of the formatting printer for the specified page format object. If the page format object does not have that information, the variable is set to `NULL`. You should not release the string without first retaining it.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Page format objects can be created a number of different ways and some of them do not require a specific printer. If the printer ID is known, the printer is displayed in the Page Setup dialog's Format for pop-up menu. If the printer ID is not known, the default formatting printer is the generic Any Printer. The printing system provides default page and paper sizes for the generic printer.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`PMCore.h`

PMPaperCreate

Creates a paper object. (Deprecated in Mac OS X v10.5. Use `PMPrinterGetPaperList` (page 2198) to find the built-in papers available for a given printer or use `PMPaperCreateCustom` (page 2177) to create a custom paper.)

```
OSStatus PMPaperCreate (
    PMPrinter printer,
    CFStringRef id,
    CFStringRef name,
    double width,
    double height,
    const PMPaperMargins *margins,
    PMPaper *paperP
);
```

Parameters

printer

A printer object for which the paper is appropriate.

id

A unique identifier for this paper.

name

The name to display to the user for this paper.

width
The width of the paper, in points.

height
The height of the paper, in points.

margins
A pointer to a `PMPaperMargins` (page 2276) structure that specifies the unprintable margins of the paper, in points. The four values in the structure specify the top, left, bottom, and right imageable area margins of the paper.

paperP
A pointer to your `PMPaper` (page 2275) variable. On return, the variable refers to a new paper object with the specified attributes. You are responsible for releasing the paper object with the function `PMRelease` (page 2214). The variable is set to `NULL` if the object could not be created.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function creates a paper object appropriate for the specified printer. To obtain one of the available built-in paper sizes for a given printer, you should use the function `PMPrinterGetPaperList` (page 2198).

Special Considerations

This function creates a paper object but does not mark it as a custom paper, so it appears to the printing system as if it were a built-in paper. This can be produce unpredictable results, so this function is deprecated.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMPaperCreateCustom

Creates a custom paper object.

```
OSStatus PMPaperCreateCustom (
    PMPrinter printer,
    CFStringRef id,
    CFStringRef name,
    double width,
    double height,
    const PMPaperMargins *margins,
    PMPaper *paperP
);
```

Parameters

printer
A printer for which the specified paper size is appropriate.

id
A unique identifier for this custom paper. For example, you could create a UUID string and use it as the unique identifier.

name
The name to display to the user for this custom paper.

width
The width of the paper, in points.

height
The height of the paper, in points.

margins
A pointer to a `PMPaperMargins` (page 2276) structure that specifies the unprintable margins of the paper, in points. The four values in the structure specify the top, left, bottom, and right imageable area margins of the paper.

paperP
A pointer to your `PMPaper` (page 2275) variable. On return, the variable refers to a new custom paper object. You are responsible for releasing the paper object with the function `PMRelease` (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function creates a custom paper object appropriate for the specified printer. Custom papers are treated differently than built-in papers by the printing system. To obtain one of the available built-in papers for a given printer, you can use the function `PMPrinterGetPaperList` (page 2198).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`PMCore.h`

PMPaperCreateLocalizedName

Obtains the localized name for a given paper.

```
OSStatus PMPaperCreateLocalizedName (
    PMPaper paper,
    PMPrinter printer,
    CFStringRef *paperName
);
```

Parameters

paper
The paper whose localized name you want to obtain.

printer
The printer for which the localization should be performed.

paperName
A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string that contains the localized name of the paper. This name is appropriate to display in the user interface. If an error occurs, the variable is set to `NULL`. You are responsible for releasing the string.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Not all printers have the same way of referring to a given paper. Generally, if you want to obtain the name of a paper, you want to localize the paper name for a particular printer. For example, if you were displaying a list of papers for a given printer, you would want the paper names to be localized for that printer.

Special Considerations

In Mac OS X v10.5 and later, Apple recommends using this function instead of `PMPaperGetName` (page 2180).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`PMCore.h`

PMPaperGetHeight

Obtains the height of the sheet of paper represented by a paper object.

```
OSStatus PMPaperGetHeight (
   PMPaper paper,
   double *paperHeight
);
```

Parameters

paper

The paper whose height you want to obtain.

paperHeight

A pointer to your double-precision variable. On return, the variable contains the height of the specified paper, in points.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPaperGetID

Obtains the identifier of a paper object.

```
OSStatus PMPaperGetID (
   PMPaper paper,
   CFStringRef *paperID
);
```

Parameters

paper

The paper whose identifier you want to obtain.

paperID

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string containing the unique identifier for this paper. You should not release the string without first retaining it.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPaperGetMargins

Obtains the margins describing the unprintable area of the sheet represented by a paper object.

```
OSStatus PMPaperGetMargins (
   PMPaper paper,
   PMPaperMargins *paperMargins
);
```

Parameters

paper

The paper whose margins you want to obtain.

paperMargins

A pointer to your `PMPaperMargins` (page 2276) structure. On return, the structure contains the unprintable margins of the specified paper, in points. The four values in the structure specify the top, left, bottom, and right imageable area margins of the paper.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPaperGetName

Obtains the name for a given paper.

```
OSStatus PMPaperGetName (
   PMPaper paper,
   CFStringRef *paperName
);
```

Parameters

paper

The paper whose name you want to obtain.

paperName

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string containing the name for this paper. This name identifies the paper in the user interface. You should not release the string without first retaining it.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Special Considerations

This function does not necessarily return a paper name that's localized for a given printer. In Mac OS X v10.5 and later, instead of using this function, Apple recommends using the function `PMPaperCreateLocalizedName` (page 2178).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPaperGetPPDPaperName

Obtains the PPD paper name for a given paper.

```
OSStatus PMPaperGetPPDPaperName (
    PMPaper paper,
    CFStringRef *paperName
);
```

Parameters

paper

The paper whose PPD paper name you want to obtain.

paperName

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string that contains the PPD paper name for the specified paper. If an error occurs, the variable is set to `NULL`. You should not release the string without first retaining it.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The Mac OS X printing system uses a PostScript Printer Description (PPD) file to describe a given printer and print queue for that printer. The PPD paper name is the name that uniquely identifies a given paper for the printer to which the paper corresponds. To obtain a list of papers for a given printer, use the function `PMPrinterGetPaperList` (page 2198).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`PMCore.h`

PMPaperGetPrinterID

Obtains the printer ID of the printer to which a given paper corresponds.

```
OSStatus PMPaperGetPrinterID (
    PMPaper paper,
    CFStringRef *printerID
);
```

Parameters

paper

The paper whose printer ID you want to obtain.

printerID

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string that contains the printer ID for the specified paper. If an error occurs, the variable is set to `NULL`. You should not release the string without first retaining it.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Not all papers have a printer ID associated with them. If the printer ID is known, the printer is displayed in the Page Setup dialog's Format for pop-up menu. If the printer ID is not known, the default formatting printer is the generic Any Printer. The printing system provides default paper sizes for the generic printer.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`PMCore.h`

PMPaperGetWidth

Obtains the width of the sheet of paper represented by a paper object.

```
OSStatus PMPaperGetWidth (
   PMPaper paper,
   double *paperWidth
);
```

Parameters

paper

The paper whose width you want to obtain.

paperWidth

A pointer to your double-precision variable. On return, the variable contains the width of the specified paper, in points.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`PMCore.h`

PMPaperIsCustom

Returns a Boolean value indicating whether a specified paper is a custom paper.

```
Boolean PMPaperIsCustom (
    PMPaper paper
);
```

Parameters

paper

The paper you're querying to determine whether it's a custom paper.

Return Value

If `true`, the specified paper is a custom paper; otherwise, `false`.

Discussion

You can create a custom paper with the function `PMPaperCreateCustom` (page 2177).

Availability

Available in Mac OS X v10.5 and later.

Declared In

`PMCore.h`

PMPostScriptBegin

Puts the current driver into PostScript mode, ready to accept PostScript data instead of QuickDraw data. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMPostScriptBegin ();
```

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Valid between calls to `PMBeginPage` and `PMEndPage`. Call `PMIsPostScriptDriver` before calling `PMPostScriptBegin` to ensure that the current driver supports PostScript data.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PMCoreDeprecated.h`

PMPostScriptData

Passes PostScript data, referenced by a pointer, to the current printer driver. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMPostScriptData (
    Ptr psPtr,
    Size len
);
```

Parameters

psPtr

A pointer to PostScript data.

len

The number of bytes of PostScript data to pass to the current driver.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid between calls to `PMPostScriptBegin` and `PMPostScriptEnd`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMPostScriptEnd

Restores the current driver to QuickDraw mode, ready to accept QuickDraw data instead of PostScript data. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMPostScriptEnd ();
```

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid between calls to `PMBeginPage` and `PMEndPage`. Call `PMPostScriptEnd` to complete a PostScript session started with `PMPostScriptBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMPostScriptFile

Passes PostScript data, contained in a file, to the current printer driver. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMPostScriptFile (
    FSSpec *psFile
);
```

Parameters

psFile

A file specification.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid between calls to `PMPostScriptBegin` and `PMPostScriptEnd`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMPostScriptHandle

Passes PostScript data, referenced by a handle, to the current printer driver. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMPostScriptHandle (
    Handle psHandle
);
```

Parameters

psHandle

A reference to PostScript data.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid between calls to `PMPostScriptBegin` and `PMPostScriptEnd`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMPresetCopyName

Obtains the localized name for a preset.

```
OSStatus PMPresetCopyName (
   PMPreset preset,
   CFStringRef *name
);
```

Parameters

preset

The preset object whose localized name you want to obtain. You can use the function `PMPrinterCopyPresets` (page 2189) to obtain the presets for a given printer.

paperID

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string containing the localized name of the specified preset. You are responsible for releasing the string.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPresetCreatePrintSettings

Creates a print settings object with settings that correspond to a preset.

```
OSStatus PMPresetCreatePrintSettings (
   PMPreset preset,
   PMPrintSession session,
   PMPrintSettings *printSettings
);
```

Parameters

preset

The preset whose settings you want to obtain. You can use the function `PMPrinterCopyPresets` (page 2189) to obtain the presets for a given printer.

session

The session you use to present the Print dialog.

printSettings

A pointer to your `PMPrintSettings` (page 2277) variable. On return, the variable refers to a print settings object with settings that correspond to the specified preset. You are responsible for releasing the print settings object with the function `PMRelease` (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPresetGetAttributes

Obtains the attributes of a preset.

```
OSStatus PMPresetGetAttributes (
   PMPreset preset,
   CFDictionaryRef *attributes
);
```

Parameters

preset

The preset whose attributes you want to obtain. You can use the function `PMPrinterCopyPresets` (page 2189) to obtain the presets for a given printer.

attributes

A pointer to your `CFDictionaryRef` variable. On return, the variable refers to a Core Foundation dictionary containing the attributes of the specified preset, or `NULL` if the attributes could not be obtained. For more information about these attributes, see the Discussion. You should not release this dictionary without first retaining it.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

A preset has associated with it a dictionary containing the preset identifier, the localized name, and a description of the environment for which the preset is intended. In addition to these standard attributes, the preset you specify may contain additional attributes that reflect custom print settings.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`PMCore.h`

PMPrinterCopyDescriptionURL

Obtains the URL of the description file for a given printer.

```
OSStatus PMPrinterCopyDescriptionURL (
   PMPrinter printer,
   CFStringRef descriptionType,
   CFURLRef *fileURL
);
```

Parameters

printer

The printer whose description file you want to obtain.

descriptionType

A constant that specifies the desired printer description file type. Currently, you must pass the constant `kPMPPDDescriptionType`.

fileURL

A pointer to your `CFURLRef` variable. On return, the variable refers to a Core Foundation URL that specifies the location of the file that contains a description of the specified printer. You are responsible for releasing the URL. If an error occurs, the variable is set to `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can use this function to locate the PostScript printer description (PPD) file for a printer.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`PMCore.h`

PMPrinterCopyDeviceURI

Obtains the device URI of a given printer.

```
OSStatus PMPrinterCopyDeviceURI (
   PMPrinter printer,
   CFURLRef *deviceURI
);
```

Parameters

printer
The printer whose device URI you want to obtain.

deviceURI
A pointer to your `CFURLRef` variable. On return, the variable refers to a Core Foundation URL that specifies the printer's device URI. You are responsible for releasing the URL. If an error occurs, the variable is set to `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The device URI of a printer describes how to communicate with the device. For some devices, it also includes a unique identifier for the device.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMPrinterCopyHostName

Obtains the name of the server hosting the print queue for a given printer.

```
OSStatus PMPrinterCopyHostName (
    PMPrinter printer,
    CFStringRef *hostNameP
);
```

Parameters

printer

The printer whose print queue host name you want to obtain.

hostNameP

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string containing the name of the specified printer's server. You are responsible for releasing the string.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

This function is typically used to obtain the name of the computer that hosts a shared printer, possibly for display in a user interface. In Mac OS X v10.5 and later, the typical way that users browse and communicate with a shared printer creates a local print queue and `PMPrinterCopyHostName` for such a print queue will return the name of the local host.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`PMCore.h`

PMPrinterCopyPresets

Obtains a list of print settings presets for a printer.

```
OSStatus PMPrinterCopyPresets (
   PMPrinter printer,
   CFArrayRef *presetList
);
```

Parameters

printer

The printer whose presets you want to obtain.

presetList

A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array containing the presets for the specified printer. Each element in the array is an object of type `PMPreset` (page 2276). You are responsible for releasing the array.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

A printer may have associated with it a list of preset settings. Each setting is optimized for a particular printing situation. This function returns all of the presets for a given printer. To obtain more information about a particular preset, you can use the function `PMPresetGetAttributes` (page 2187). To create a print settings object that contains the settings of a preset, call `PMPresetCreatePrintSettings` (page 2186).

Availability

Available in Mac OS X v10.3 and later.

Declared In
PMCore.h

PMPrinterCreateFromPrinterID

Creates a printer object from a print queue identifier.

```
PMPrinter PMPrinterCreateFromPrinterID (
   CFStringRef printerID
);
```

Parameters

printerID
The unique identifier of a print queue.

Return Value
A new printer object, or NULL if no print queue is available with the specified identifier. You are responsible for releasing the printer object with the function PMRelease (page 2214).

Discussion
This function is typically used to re-create a printer object using the print queue ID obtained by a call to PMPrinterGetID at an earlier time. If the print queue is deleted after obtaining the ID, this function returns NULL for that ID.

Availability
Available in Mac OS X v10.4 and later.

See Also
PMPrinterGetID (page 2193)

Declared In
PMCore.h

PMPrinterGetCommInfo

Obtains information about the communication channel for a printer.

```
OSStatus PMPrinterGetCommInfo (
   PMPrinter printer,
   Boolean *supportsTransparentP,
   Boolean *supportsEightBitP
);
```

Parameters

printer
The printer whose information you want to obtain.

supportsTransparentP
A pointer to your Boolean variable. On return, true indicates that the communication channel to the specified printer supports bytes in the range 0x0–0x1F; otherwise, false.

supportsEightBitP
A pointer to your Boolean variable. On return, true indicates that the communication channel to the specified printer supports bytes in the range 0x80–0xFF; otherwise, false.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is typically relevant only to PostScript printers. All PostScript printers, regardless of what communications channel is used to send data to them, support data in the range 0x20–0x7F. Many communications channels can support data outside this range. You can use this function to determine whether the communications channel to the specified printer also supports bytes in the ranges 0x0–0x1F and 0x80–0xFF.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPrinterGetDescriptionURL

Obtains a reference to the specified printer's description file. (Deprecated in Mac OS X v10.4. Use `PMPrinterCopyDescriptionURL` (page 2187) instead.)

```
OSStatus PMPrinterGetDescriptionURL (
   PMPrinter printer,
   CFStringRef descriptionType,
   CFURLRef *fileURL
);
```

Parameters

printer
The printer whose description file you want to obtain.

descriptionType
A Core Foundation string that specifies the type of description file for the selected printer. Currently, there is only one type defined—`kPMPPDDescriptionType`.

fileURL
A pointer to your `CFURLRef` variable. On return, the variable refers to a URL for the printer's description file. In spite of the name, the function `PMPrinterGetDescriptionURL` has Create/Copy semantics which means the caller must release the returned URL if it is not `NULL` and the result code `noErr` is returned.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can use this function to obtain a reference to the PostScript printer description (PPD) file for a PostScript printer.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMPrinterGetDeviceURI

Obtains a copy of a printer's device URI. (Deprecated in Mac OS X v10.4. Use `PMPrinterCopyDeviceURI` (page 2188) instead.)

```
OSStatus PMPrinterGetDeviceURI (
   PMPrinter printer,
   CFURLRef *deviceURI
);
```

Parameters

printer

The printer whose device URI you want to obtain.

deviceURI

A pointer to your `CFURLRef` variable. On return, the variable refers to a URI for the location of the printer device. In spite of the name, this function has Create/Copy semantics which means the caller must release the returned URL if it is not `NULL` and the result code `noErr` is returned.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMPrinterGetDriverCreator

Obtains the creator of the driver associated with the specified printer.

```
OSStatus PMPrinterGetDriverCreator (
   PMPrinter printer,
   OSType *creator
);
```

Parameters

printer

The printer whose driver creator you want to obtain.

creator

On return, the 4-byte creator code of the driver (for example, `'APPL'` for an Apple printer driver).

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

This function is not recommended because it makes your application driver-dependent.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PMCore.h`

PMPrinterGetDriverReleaseInfo

Obtains version information for the driver associated with the specified printer.

```
OSStatus PMPrinterGetDriverReleaseInfo (
   PMPrinter printer,
   VersRec *release
);
```

Parameters

printer

The printer whose driver version you want to obtain.

release

A pointer to your `VersRec` data structure. On return, the structure contains the driver's short and long version strings and country code.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

This function is not recommended because it makes your application driver-dependent. If you do use this function, you must call it between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PMCore.h`

PMPrinterGetID

Returns the unique identifier of a printer.

```
CFStringRef PMPrinterGetID (
   PMPrinter printer
);
```

Parameters

printer

The printer whose identifier you want to obtain.

Return Value

The identifier of the specified printer. You should not release the string without first retaining it. If the specified printer is not valid, this function returns `NULL`.

Discussion

You can use the function `PMPrinterGetID` to capture information about a printer for later use. To create a printer object from a printer ID returned by this function, use the function `PMPrinterCreateFromPrinterID` (page 2190).

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

PMPrinterTest

Declared In
PMCore.h

PMPrinterGetIndexedPrinterResolution

Obtains a resolution setting based on an index into the range of settings supported by the specified printer.

```
OSStatus PMPrinterGetIndexedPrinterResolution (
   PMPrinter printer,
   UInt32 index,
   PMResolution *resolutionP
);
```

Parameters

printer
The printer whose resolution you want to obtain.

index
An index into the range of resolution settings supported by the specified printer. Index values begin at 1.

res
A pointer to your PMResolution (page 2278) data structure. On return, the structure contains the printer resolution setting associated with the index value.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. Before you call this function, you must call the function PMPrinterGetPrinterResolutionCount (page 2199) to obtain the number of resolution settings supported by the specified printer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
PMCore.h

PMPrinterGetLanguageInfo

Obtains information about the imaging language for the specified printer.

```
OSStatus PMPrinterGetLanguageInfo (
   PMPrinter printer,
   PMLanguageInfo *info
);
```

Parameters

printer
The printer whose imaging language information you want to obtain.

info

A pointer to your `PMLanguageInfo` (page 2274) data structure. On return, the structure contains the printer's language level, version, and release information. The format of the returned data uses the syntax of the PostScript language.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The function `PMPrinterGetLanguageInfo` is useful only for PostScript printers. You must call this function between the creation and release of a printing session.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMCore.h`

PMPrinterGetLocation

Returns the location of a printer.

```
CFStringRef PMPrinterGetLocation (
   PMPrinter printer
);
```

Parameters

printer

The printer whose location you want to obtain.

Return Value
The location of the specified printer. You should not release the string without first retaining it. If the printer is not valid, this function returns `NULL`.

Discussion
The location of a printer is specified when a user creates a print queue for the printer. In some cases, the printing system automatically determines the location. For example, the location may be set to "Local Zone". The user creating the print queue can also set the location.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMPrinterGetMakeAndModelName

Obtains the manufacturer and model name of the specified printer.

```
OSStatus PMPrinterGetMakeAndModelName (
   PMPrinter printer,
   CFStringRef *makeAndModel
);
```

Parameters

printer

The printer whose manufacturer and model name you want to obtain.

makeAndModel

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string containing the manufacturer and model name of the specified printer. You should not release the string without first retaining it. If an error occurs, the variable is set to `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMPrinterGetMimeTypes

Obtains a list of MIME content types supported by a printer using the specified print settings.

```
OSStatus PMPrinterGetMimeTypes (
   PMPrinter printer,
   PMPrintSettings settings,
   CFArrayRef *mimeTypes
);
```

Parameters

printer

The printer whose supported MIME types you want to obtain.

settings

The print settings for the print job. The print settings object contains the job destination, which affects the available types. This parameter may be `NULL`.

mimeTypes

A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array containing the MIME types supported by the specified printer. Each element in the array is a Core Foundation string. You should not release the array without first retaining it.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function retrieves the types of data that can be submitted to a printer with the specified print settings; for example, `application/pdf`. This function is typically used in conjunction with the function `PMPrinterPrintWithFile` (page 2203).

Availability
Available in Mac OS X v10.3 and later.

Declared In
PMCore.h

PMPrinterGetName

Returns the human-readable name of a printer.

```
CFStringRef PMPrinterGetName (
   PMPrinter printer
);
```

Parameters

printer

The printer whose name you want to obtain.

Return Value
The name of the specified printer. This name identifies the printer in the user interface. You should not release the string without first retaining it.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
PMPrinterTest

Declared In
PMCore.h

PMPrinterGetOutputResolution

Obtains the printer hardware output resolution for the specified print settings.

```
OSStatus PMPrinterGetOutputResolution (
    PMPrinter printer,
    PMPrintSettings printSettings,
    PMResolution *resolutionP
);
```

Parameters

printer

The printer whose output resolution you want to obtain.

printSettings

The print settings you want to use.

resolutionP

A pointer to your PMResolution (page 2278) structure. On return, the structure contains the output resolution of the specified printer in pixels per inch.

Return Value
A result code. If the resolution cannot be reliably determined, this function returns an error.

Discussion
Some printers allow programmatic control of their hardware output resolution on a print job basis. The hardware resolution is determined by the combination of printer and print settings used for the print job. This function returns the best guess as to what printer resolution setting will be used for the destination print job.

Most applications do not need to use this function because they draw the same content regardless of the destination device. For those few applications that do adjust their drawing based on the output device, they should only do so when the print job destination is `kPMDestinationPrinter` or `kPMDestinationFax`. You can use the function `PMSessionGetDestinationType` to determine the destination for a print job.

This function should be used after displaying the Print dialog to the user so that it correctly reflects changes in print settings performed prior to printing.

Availability
Available in Mac OS X v10.5 and later.

See Also
`PMPrinterSetOutputResolution` (page 2205)

Declared In
`PMCore.h`

PMPrinterGetPaperList

Obtains the list of papers available for a printer.

```
OSStatus PMPrinterGetPaperList (
   PMPrinter printer,
   CFArrayRef *paperList
);
```

Parameters

printer
The printer whose paper list you want to obtain.

paperList
A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array containing the paper list for the specified printer. Each element in the array is an object of type `PMPaper` (page 2275). You should not release the array without first retaining it.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function obtains a list of the papers that a given printer claims to support. The paper list does not include any custom paper sizes that may be available.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPrinterGetPrinterResolution

Obtains a resolution setting for the specified printer. (Deprecated in Mac OS X v10.5. Use `PMPrinterGetPrinterResolutionCount` (page 2199) and `PMPrinterGetIndexedPrinterResolution` (page 2194) to examine the available printer resolutions.)

```
OSStatus PMPrinterGetPrinterResolution (
   PMPrinter printer,
   PMTag tag,
   PMResolution *res
);
```

Parameters

printer

The printer whose resolution you want to obtain.

tag

A tag that specifies the kind of resolution information you want to obtain (minimum, maximum, default, and so forth). See "Tag Constants" (page 2295) for a description of the constants you can pass in this parameter.

res

A pointer to your `PMResolution` (page 2278) data structure. On return, the structure contains the resolution setting associated with the tag value.

Return Value

A result code. The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMPrinterGetPrinterResolutionCount

Obtains the number of resolution settings supported by the specified printer.

```
OSStatus PMPrinterGetPrinterResolutionCount (
   PMPrinter printer,
   UInt32 *countP
);
```

Parameters

printer

The printer whose number of resolution settings you want to obtain.

count

A pointer to your `UInt32` variable. On return, the variable contains the number of resolutions that are supported for the specified printer.

Return Value

A result code. The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMCore.h`

PMPrinterGetState

Obtains the current state of the print queue for a printer.

```
OSStatus PMPrinterGetState (
   PMPrinter printer,
   PMPrinterState *state
);
```

Parameters

printer

The printer whose queue state you want to obtain.

state

A pointer to your `PMPrinterState` variable. On return, the variable contains a constant that indicates the current state of the print queue for the specified printer. Supported values are:

- `kPMPrinterIdle` (queue is idle)
- `kPMPrinterProcessing` (queue is processing a job)
- `kPMPrinterStopped` (queue is stopped)

See "Print Queue States" (page 2294) for a complete description of these constants.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMPrinterIsDefault

Returns a Boolean value indicating whether a printer is the default printer for the current user.

```
Boolean PMPrinterIsDefault (
   PMPrinter printer
);
```

Parameters

printer

The printer you're querying to determine whether it is the default printer.

Return Value
If `true`, the specified printer is the default printer for the current user; otherwise, `false`.

Discussion
The default printer is the printer selected by default in the Print dialog.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMPrinterIsFavorite

Returns a Boolean value indicating whether a printer is in the user's list of favorite printers.

```
Boolean PMPrinterIsFavorite (
   PMPrinter printer
);
```

Parameters

printer
The printer you're looking for in the favorite printer list.

Return Value
If `true`, the specified printer is in the user's list of favorite printers; otherwise, `false`.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMPrinterIsPostScriptCapable

Returns a Boolean value indicating whether a printer is PostScript capable.

```
Boolean PMPrinterIsPostScriptCapable (
   PMPrinter printer
);
```

Parameters

printer
The printer you're querying to determine whether it's PostScript capable.

Return Value
If `true`, the specified printer is a PostScript capable printer; otherwise, `false`.

Discussion
A printer that is PostScript capable is not necessarily a PostScript printer. The Mac OS X printing system can render PostScript content on non-PostScript printers.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`PMCore.h`

PMPrinterIsPostScriptPrinter

Determines whether a printer is a PostScript printer.

```
OSStatus PMPrinterIsPostScriptPrinter (
    PMPrinter printer,
    Boolean *isPSPrinter
);
```

Parameters

printer
The printer you're querying to determine whether it's a PostScript printer.

isPSPrinter
A pointer to your Boolean variable. On return, `true` indicates that the specified printer is a PostScript printer; otherwise, `false`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
A printer is a PostScript printer if the printer driver takes PostScript directly.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`PMCore.h`

PMPrinterIsRemote

Indicates whether a printer is hosted by a remote print server.

```
OSStatus PMPrinterIsRemote (
    PMPrinter printer,
    Boolean *isRemoteP
);
```

Parameters

printer
The printer you're querying to determine whether it is hosted by a remote print server.

isRemoteP
A pointer to your Boolean variable. On return, `true` indicates that the printer is hosted by a remote print server; otherwise, `false`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
If this function returns `true`, the printer is hosted by a remote print server and the printer can be considered a shared printer.

In Mac OS X, the typical way that users create a print queue for a shared printer is by browsing. Print queues for shared printers that are created by browsing are marked as remote queues, and `PMPrinterIsRemote` returns `true` for such printers. However, expert users can create a local queue for a remote printer manually, and such a printer does not appear to be remote printer.

Whether a printer is remote is derived from the CUPS printer-type attribute for the print queue.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMPrinterPrintWithFile

Submits a print job to a specified printer using a file that contains print data.

```
OSStatus PMPrinterPrintWithFile (
   PMPrinter printer,
   PMPrintSettings settings,
   PMPageFormat format,
   CFStringRef mimeType,
   CFURLRef fileURL
);
```

Parameters

printer
The destination printer.

settings
The print settings for the print job.

format
The physical page size and orientation with which the document should be printed. This parameter can be `NULL`.

mimeType
The MIME type of the data to be printed. If this parameter is `NULL`, the MIME type will be determined automatically. You can obtain a list of the MIME types supported by a given printer using the function `PMPrinterGetMimeTypes` (page 2196).

fileURL
The URL of the file that supplies the print data.

Return Value
A result code. See "Core Printing Result Codes" (page 2298). If the specified printer cannot handle the file's MIME type, a non-zero error code is returned.

Discussion
This function can fail if the specified printer cannot handle the file's MIME type. Use the function `PMPrinterGetMimeTypes` (page 2196) to check whether a MIME type is supported.

Availability
Available in Mac OS X v10.3 and later.

See Also
`PMPrinterPrintWithProvider` (page 2204)

Declared In
`PMCore.h`

PMPrinterPrintWithProvider

Submits a print job to a specified printer using a Quartz data provider to obtain the print data.

```
OSStatus PMPrinterPrintWithProvider (
   PMPrinter printer,
   PMPrintSettings settings,
   PMPageFormat format,
   CFStringRef mimeType,
   CGDataProviderRef provider
);
```

Parameters

printer
The destination printer.

settings
The print settings for the print job.

format
The physical page size and orientation with which the document should be printed. This parameter can be `NULL`.

mimeType
The MIME type of the data to be printed. This parameter cannot be `NULL`. If you want automatic typing, use the function `PMPrinterPrintWithFile` (page 2203) instead. You can obtain a list of the MIME types supported by a given printer using the function `PMPrinterGetMimeTypes` (page 2196).

provider
The data provider that supplies the print data.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function can fail if the specified printer cannot handle the data provider's MIME type. Use the function `PMPrinterGetMimeTypes` (page 2196) to check whether a MIME type is supported.

Special Considerations

In Mac OS X v10.4 and earlier, this function is not implemented and returns the error code –1 when called. You can write your print data to a file and use `PMPrinterPrintWithFile` instead.

Availability
Available in Mac OS X v10.3 and later.

See Also
PMPrinterPrintWithFile (page 2203)

Declared In
PMCore.h

PMPrinterSetDefault

Sets the default printer for the current user.

```
OSStatus PMPrinterSetDefault (
    PMPrinter printer
);
```

Parameters

printer
The printer to set as the default printer.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The default printer is the printer selected by default in the Print dialog.

This function is rarely used. Most applications do not set the default printer directly, but instead let the user choose the default printer in the Print & Fax preference pane of System Preferences.

Availability
Available in Mac OS X v10.5 and later.

Declared In
PMCore.h

PMPrinterSetOutputResolution

Sets the print settings to reflect the specified printer hardware output resolution.

```
OSStatus PMPrinterSetOutputResolution (
    PMPrinter printer,
    PMPrintSettings printSettings,
    const PMResolution *resolutionP
);
```

Parameters

printer
The printer whose output resolution you want to change.

printSettings
The print settings object used for the print job.

resolutionP
A pointer to a PMResolution (page 2278) structure that specifies the desired resolution in pixels per inch.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Some printers allow programmatic control of their hardware output resolution on a print job basis. The hardware resolution is determined by the combination of printer and print settings used for the print job. This function configures the print settings to the closest resolution setting that can be used for the destination print job. Note that not all printers allow control of their resolution setting.

This function is rarely used. Most applications do not set the output resolution but instead use the setting supplied by the user in the Print dialog.

Availability
Available in Mac OS X v10.5 and later.

See Also
`PMPrinterGetOutputResolution` (page 2197)

Declared In
`PMCore.h`

PMPrinterWritePostScriptToURL

Converts an input file of the specified MIME type to printer-ready PostScript for a destination printer.

```
OSStatus PMPrinterWritePostScriptToURL (
    PMPrinter printer,
    PMPrintSettings settings,
    PMPageFormat format,
    CFStringRef mimeType,
    CFURLRef sourceFileURL,
    CFURLRef destinationFileURL
);
```

Parameters

printer
The destination printer for which printer-ready PostScript will be generated.

settings
The print settings for the print job.

format
The page format specifying the physical page size and orientation on which the document should be printed.

mimeType
The MIME type of the file to be printed. If you pass `NULL`, the file is typed automatically. You can obtain a list of the MIME types supported by a given printer using the function `PMPrinterGetMimeTypes` (page 2196).

sourceFileURL
A URL specifying the input file to be converted to printer-ready PostScript data. Only file-based URLs are supported.

destinationFileURL
A URL specifying the destination file to be created. If the file already exists, it will be overwritten. Only file-based URLs are supported.

Return Value
A result code. If the printing system cannot convert the input MIME type to PostScript, this function fails and returns an error.

Discussion
This function is synchronous; the conversion of the input file to PostScript is performed before the function returns. This can take a significant amount of time for longer documents. You may want to perform this operation on a thread other than the main application thread or fork a separate process for this purpose.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`PMCore.h`

PMPrintSettingsCopyAsDictionary

Creates a dictionary that contains the settings in a print settings object.

```
OSStatus PMPrintSettingsCopyAsDictionary (
    PMPrintSettings printSettings,
    CFDictionaryRef *settingsDictionary
);
```

Parameters

printSettings
The print settings object with the desired settings.

settingsDictionary
A pointer to your `CFDictionaryRef` variable. On return, the variable refers to a Core Foundation dictionary that contains the settings in the specified print settings object. Some of the keys in this dictionary are currently defined in `PMTicket.h`; other keys are user-defined. You are responsible for releasing the dictionary. If an error occurs, the variable is set to `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Most developers have no need to use this function. However, one way this function might be useful would be to enumerate all the entries in a print settings object for inspection.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`PMCore.h`

PMPrintSettingsCopyKeys

Obtains the keys for items in a print settings object.

```
OSStatus PMPrintSettingsCopyKeys (
    PMPrintSettings printSettings,
    CFArrayRef *settingsKeys
);
```

Parameters

printSettings

The print settings object with the desired keys.

settingsKeys

A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array that contains the keys for items in the specified print settings object. Each of these keys may be passed to the function `PMPrintSettingsGetValue` to obtain a value. You are responsible for releasing the array. If an error occurs, the variable is set to `NULL`.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

This function provides an array of the keys in a print settings object. You could get the values for the keys in the array with `PMPrintSettingsGetValue` (page 2210), or use the keys to look up the values in the dictionary returned by `PMPrintSettingsCopyAsDictionary` (page 2207).

Availability

Available in Mac OS X v10.5 and later.

Declared In

`PMCore.h`

PMPrintSettingsCreateDataRepresentation

Creates a data representation of a print settings object.

```
OSStatus PMPrintSettingsCreateDataRepresentation (
    PMPrintSettings printSettings,
    CFDataRef *data,
    PMDataFormat format
);
```

Parameters

printSettings

The print settings object to convert.

data

A pointer to your `CFDataRef` variable. On return, the variable refers to a new Core Foundation data object that contains a representation of the specified print settings object in the specified data format. You are responsible for releasing the data object.

format

A constant that specifies the format of the data representation. Supported values are:

- `kPMDataFormatXMLDefault` (compatible with all Mac OS X versions)
- `kPMDataFormatXMLMinimal` (approximately 3-5 times smaller; compatible with Mac OS X v10.5 and later)
- `kPMDataFormatXMLCompressed` (approximately 20 times smaller; compatible with Mac OS X v10.5 and later)

See "Data Representation Formats" (page 2280) for a full description of these formats.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is typically used to convert a print settings object into a data representation suitable for storage in a user document. For information about using a Core Foundation data object, see *CFData Reference*.

Before calling this function, you should call the function `PMSessionValidatePrintSettings` (page 2249) to make sure the print settings object contains valid values.

Apple recommends that you do not reuse the print settings information if the user prints the document again. The information supplied by the user in the Print dialog should pertain to the document only while the document prints, so there is no need to save the print settings object.

Availability
Available in Mac OS X v10.5 and later.

See Also
`PMPrintSettingsCreateWithDataRepresentation` (page 2209)

Declared In
`PMCore.h`

PMPrintSettingsCreateWithDataRepresentation

Creates a print settings object from a data representation.

```
OSStatus PMPrintSettingsCreateWithDataRepresentation (
    CFDataRef data,
    PMPrintSettings *printSettings
);
```

Parameters

data

The data representation of a print settings object. The data representation must have been previously created with the function `PMPrintSettingsCreateDataRepresentation` (page 2208).

printSettings

A pointer to your `PMPrintSettings` (page 2277) variable. On return, the variable refers to a new print settings object that contains the printing information stored in the specified data object. You are responsible for releasing the print settings object with the function `PMRelease` (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is typically used to convert a data representation stored in a user document back into a print settings object. For information about creating a Core Foundation data object from raw data, see *CFData Reference*.

After calling this function, you should call the function `PMSessionValidatePrintSettings` (page 2249) to make sure the print settings object contains valid values.

Availability
Available in Mac OS X v10.5 and later.

See Also
`PMPrintSettingsCreateDataRepresentation` (page 2208)

Declared In
`PMCore.h`

PMPrintSettingsGetJobName

Obtains the name of a print job.

```
OSStatus PMPrintSettingsGetJobName (
   PMPrintSettings printSettings,
   CFStringRef *name
);
```

Parameters

printSettings
The print settings for the current print job.

name
A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string containing the name of the print job. This is the same job name you set using the function `PMPrintSettingsSetJobName` (page 2211). You should not release the string without first retaining it.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.

See Also
`PMPrintSettingsSetJobName` (page 2211)

Declared In
`PMCore.h`

PMPrintSettingsGetValue

Obtains the value of a setting in a print settings object.

```
OSStatus PMPrintSettingsGetValue (
    PMPrintSettings printSettings,
    CFStringRef key,
    CFTypeRef *value
);
```

Parameters

printSettings

The print settings object you want to access.

key

A string constant that specifies the key for the desired setting. Some keys are currently defined in `PMTicket.h`; other keys are user-defined.

value

A pointer to your Core Foundation variable. On return, the variable refers to a Core Foundation object that corresponds to the specified key. If no corresponding object exists, the variable is set to `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function, together with the function `PMPrintSettingsSetValue`, makes it possible to access print settings directly.

Availability
Available in Mac OS X v10.4 and later.

See Also
`PMPrintSettingsSetValue` (page 2212)

Declared In
`PMCore.h`

PMPrintSettingsSetJobName

Specifies the name of a print job.

```
OSStatus PMPrintSettingsSetJobName (
   PMPrintSettings printSettings,
   CFStringRef name
);
```

Parameters

printSettings

The print settings object whose job name you want to set.

name

The new name for the print job.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
If you're using the Print dialog, you should call this function before presenting the dialog. You are strongly encouraged to create a print job name that's meaningful to the user and use this function to set the name; this produces the best user experience. If you do not specify the print job name, the printing system creates an appropriate job name for you.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.4 and later.

See Also
`PMPrintSettingsGetJobName` (page 2210)

Declared In
`PMCore.h`

PMPrintSettingsSetValue

Stores the value of a setting in a print settings object.

```
OSStatus PMPrintSettingsSetValue (
    PMPrintSettings printSettings,
    CFStringRef key,
    CFTypeRef value,
    Boolean locked
);
```

Parameters

printSettings
The print settings object you want to update.

key
A string constant that specifies the key for the desired setting. Some keys are currently defined in `PMTicket.h`; other keys are user-defined.

value
A Core Foundation object that corresponds to the specified key. If you pass `NULL`, any existing setting for the specified key is removed.

locked
If `true`, the item being set should be locked; otherwise, `false`. Currently, you should always pass `false`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function makes it possible to add, change, or remove print settings directly. Print settings are stored as key-value pairs. The keys are Core Foundation strings and the corresponding values are Core Foundation objects.

You can use this function to store user-defined data in a print settings object. You should make sure that the custom keys you define for your private data do not conflict with any other keys in the object. Each data item you store needs to be a Core Foundation object. You can use the function `PMPrintSettingsGetValue` to retrieve your private data.

If you call this function after initiating a print job (for example, by calling `PMSessionBeginCGDocument`), the change is ignored for the current job.

Availability
Available in Mac OS X v10.4 and later.

See Also
`PMPrintSettingsGetValue` (page 2210)

Declared In
`PMCore.h`

PMPrintSettingsToOptions

Converts print settings into a CUPS options string.

```
OSStatus PMPrintSettingsToOptions (
    PMPrintSettings settings,
    char **options
);
```

Parameters

settings
The print settings to convert.

options
A pointer to a C string. On return, a CUPS options string describing the print settings, or `NULL` if the print settings could not be converted. The function allocates storage for the string. You are responsible for freeing the storage.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function creates a CUPS options string that captures the data in the specified print settings object. In Mac OS X v10.5 and later, Apple recommends that you use the `PMPrintSettingsToOptionsWithPrinterAndPageFormat` (page 2213) function instead.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

PMPrintSettingsToOptionsWithPrinterAndPageFormat

Converts print settings and page format data into a CUPS options string for a specified printer.

```
OSStatus PMPrintSettingsToOptionsWithPrinterAndPageFormat (
    PMPrintSettings settings,
    PMPrinter printer,
    PMPageFormat pageFormat,
    char **options
);
```

Parameters

settings

The print settings to convert.

printer

The printer to use for converting the print settings. This parameter must not be `NULL`.

pageFormat

The page format to convert, or `NULL` to specify default page format data.

options

A pointer to a C string. On return, a CUPS option string with the specified print settings and page format data, or `NULL` if the data could not be converted. The function allocates storage for the string. You are responsible for freeing the storage.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

This function creates a CUPS options string for the destination printer that captures the data in the specified print settings and page format objects. For example, you could pass this string to the function `PMWorkflowSubmitPDFWithOptions` (page 2271) to submit a PDF file for workflow processing. You could also use the options string to run a CUPS filter directly.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`PMCore.h`

PMRelease

Releases a printing object by decrementing its reference count.

```
OSStatus PMRelease (
   PMObject object
);
```

Parameters

object

The printing object you want to release.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Your application should use the `PMRelease` function to release any printing objects it creates or retains. When an object's reference count reaches 0, the object is deallocated.

For example, to terminate a printing session created with the function `PMCreateSession` (page 2145), pass the associated `PMPrintSession` (page 2277) object to `PMRelease`. To release printing objects created with the functions `PMCreatePageFormat` (page 2143) and `PMCreatePrintSettings` (page 2144), pass the associated `PMPageFormat` and `PMPrintSettings` objects to `PMRelease`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMRetain` (page 2215)

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMRetain

Retains a printing object by incrementing its reference count.

```
OSStatus PMRetain (
   PMObject object
);
```

Parameters

object
The printing object you want to retain.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You should retain a printing object when you receive it from elsewhere (that is, you did not create or copy it) and you want it to persist. If you retain a printing object, you are responsible for releasing it. (See `PMRelease`.) You can use the function `PMRetain` to increment a printing object's reference count so that multiple threads or routines can use the object without the risk of another thread or routine deallocating the object.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMRelease` (page 2214)

Declared In
`PMCore.h`

PMServerCreatePrinterList

Creates a list of printers available to a print server.

```
OSStatus PMServerCreatePrinterList (
   PMServer server,
   CFArrayRef *printerList
);
```

Parameters

server

The print server whose printers you want to obtain. To specify the local print server, pass the constant `kPMServerLocal`. Currently, you may specify only the local print server.

printerList

A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array containing the printers available to the specified print server. Each element in the array is a `PMPrinter` object. You are responsible for releasing the array.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
PMPrinterTest

Declared In
`PMCore.h`

PMServerLaunchPrinterBrowser

Launches the printer browser to browse the printers available for a print server.

```
OSStatus PMServerLaunchPrinterBrowser (
    PMServer server,
    CFDictionaryRef options
);
```

Parameters

server

The print server to browse. Pass `kPMServerLocal` to specify the local print server. Currently, you may specify only the local print server.

options

This parameter is reserved for future use. At the present time, pass `NULL`. Passing `NULL` presents the printer browser in the default fashion.

Return Value
A result code. See "Core Printing Result Codes" (page 2298). If you specify a server whose printers cannot be browsed, this function returns the error code `kPMInvalidParameter`.

Discussion
This function displays the standard printer browser to allow the user to create a new print queue.

Availability
Available in Mac OS X v10.5 and later.

Declared In
PMCore.h

PMSessionBeginCGDocumentNoDialog

Begins a print job that draws into a Quartz graphics context and suppresses the printing status dialog.

```
OSStatus PMSessionBeginCGDocumentNoDialog (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat pageFormat
);
```

Parameters

printSession
The printing session that provides a context for the new print job.

printSettings
The print settings to use for the new print job.

pageFormat
The page format to use for the new print job.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function starts a print job that draws directly into a Quartz graphics context and should be called within your application's print loop. This function is similar to the function PMSessionBeginCGDocument except that the printing status dialog is suppressed.

You must call PMSessionBeginCGDocumentNoDialog between the creation and release of a printing session. See the function PMCreateSession (page 2145). If you present a printing dialog before you call PMSessionBeginCGDocumentNoDialog, when calling this function you should use the same PMPrintSession (page 2277) object you used to present the dialog.

Before you call PMSessionBeginCGDocumentNoDialog, you should call PMSessionValidatePrintSettings (page 2249) and PMSessionValidatePageFormat (page 2248) to make sure the specified print settings and page format objects are updated and valid. After you call PMSessionBeginCGDocumentNoDialog, if you call a function that changes the specified print settings or page format object, the change is ignored for the current print job.

During the print job, the caller cannot obtain a Quickdraw graphics port for the printing session but can only obtain a Quartz graphics context. As a result, this function should be used in conjunction with PMSessionGetCGGraphicsContext (page 2230) instead of PMSessionGetGraphicsContext (page 2234).

This function must be called before its corresponding End function (PMSessionEndDocumentNoDialog (page 2227)). If the function PMSessionBeginCGDocumentNoDialog returns noErr, you must later call the End function, even if errors occur within the scope of the Begin and End functions.

The printing system automatically handles printing multiple copies. Your application does not need to perform any tasks other than specifying the number of copies in the printing session.

Availability
Available in Mac OS X v10.4 and later.

Declared In
PMCore.h

PMSessionBeginDocumentNoDialog

Begins a print job that, by default, draws into a QuickDraw graphics port, and suppresses the printing status dialog. (Deprecated in Mac OS X v10.5. Use PMSessionBeginCGDocumentNoDialog (page 2217) instead.)

```
OSStatus PMSessionBeginDocumentNoDialog (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat pageFormat
);
```

Parameters

printSession
The printing session that provides a context for the new print job.

printSettings
The print settings to use for the new print job.

pageFormat
The page format to use for the new print job.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The function PMSessionBeginDocumentNoDialog starts a print job and should be called within your application's print loop. This function is similar to the function PMSessionBeginDocument except that the printing status dialog is suppressed.

You must call this function between the creation and release of a printing session. See the function PMCreateSession (page 2145). If you present a printing dialog before you call PMSessionBeginDocumentNoDialog, when calling this function you should use the same PMPrintSession (page 2277) object you used to present the dialog.

Before you call PMSessionBeginDocumentNoDialog, you should call PMSessionValidatePrintSettings (page 2249) and PMSessionValidatePageFormat (page 2248) to make sure the specified print settings and page format objects are updated and valid. After you call PMSessionBeginDocumentNoDialog, if you call a function that changes the specified print settings or page format object, the change is ignored for the current print job.

This function must be called before its corresponding End function (PMSessionEndDocumentNoDialog (page 2227)). If the function PMSessionBeginDocumentNoDialog returns noErr, you must call the End function, even if errors occur within the scope of the Begin and End functions.

The printing system automatically handles printing multiple copies. Your application does not need to perform any tasks other than specifying the number of copies in the printing session.

Special Considerations

In Mac OS X v10.4 and later, Apple recommends using the function PMSessionBeginCGDocumentNoDialog (page 2217) instead of this function. QuickDraw is deprecated and your application should be using Quartz 2D for its rendering.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionBeginPageNoDialog

Starts a new page for printing in the specified printing session and suppresses the printing status dialog.

```
OSStatus PMSessionBeginPageNoDialog (
    PMPrintSession printSession,
    PMPageFormat pageFormat,
    const PMRect *pageFrame
);
```

Parameters

printSession
The printing session that provides a context for the print job.

pageFormat
The page format for the new page. If you pass `NULL`, the printing system uses the page format you passed to `PMSessionBeginCGDocumentNoDialog` (page 2217).

pageFrame
You should pass `NULL`, as this parameter is currently unsupported.

Return Value
A result code. If the user cancels the print job, this function returns `kPMCancel`.

Discussion
This function is similar to the function `PMSessionBeginPage` except that the function `PMSessionBeginPageNoDialog` suppresses the printing status dialog. You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must call the functions `PMSessionBeginPageNoDialog` and `PMSessionEndPageNoDialog` (page 2228) within the scope of calls to the `Begin` print job function (`PMSessionBeginCGDocumentNoDialog` (page 2217)) and the `End` print job function (`PMSessionEndDocumentNoDialog` (page 2227)).

You should call the function `PMSessionError` (page 2229) immediately before you call `PMSessionBeginPageNoDialog`. If `PMSessionError` returns an error, then you should not call the function `PMSessionBeginPageNoDialog`. Because `PMSessionBeginPage` also initializes the printing graphics context, your application should not make assumptions about the state of the context (for example, the current font) between successive pages. After each call to `PMSessionBeginPageNoDialog`, your application should call `PMSessionGetCGGraphicsContext` (page 2230) to obtain the current printing context.

If the function `PMSessionBeginPageNoDialog` returns `noErr`, you must later call the function `PMSessionEndPageNoDialog`, even if errors occur within the scope of `PMSessionBeginPageNoDialog` and `PMSessionEndPageNoDialog`.

The printing system automatically handles printing multiple copies. Your application does not need to perform any tasks other than specifying the number of copies in the printing session.

Special Considerations

Prior to Mac OS X v10.5, the *pageFormat* parameter is ignored. In Mac OS X v10.5 and later, the printing system supports multiple orientations within a print job. When you call this function and supply a page format, the orientation specified in the page format is used for the current page. Other settings in the page format, such as paper size or scaling, are ignored.

Availability

Available in Mac OS X v10.2 and later.

Declared In

PMCore.h

PMSessionConvertOldPrintRecord

Creates new page format and print settings objects from an old-style print record created for the classic Printing Manager. (Deprecated in Mac OS X v10.4. There is no replacement; during the transition from Mac OS 9 to Mac OS X, this function facilitated the migration of print records saved in documents created in Mac OS 9, but the function no longer serves any useful purpose in Mac OS X.)

```
OSStatus PMSessionConvertOldPrintRecord (
   PMPrintSession printSession,
   Handle printRecordHandle,
   PMPrintSettings *printSettings,
   PMPageFormat *pageFormat
);
```

Parameters

printSession

The current printing session.

printRecordHandle

A handle to an old-style print record created by the classic Printing Manager. You are responsible for disposing of the handle.

printSettings

On return, a print settings object that contains values converted from the print record. You are responsible for releasing the print settings object with the function PMRelease (page 2214).

pageFormat

On return, a page format object that contains values converted from the print record. You are responsible for releasing the page format object with the function PMRelease (page 2214).

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function PMCreateSession (page 2145).

You can use PMSessionConvertOldPrintRecord to create page format and print settings objects from old-style print records stored in documents created by pre-Carbon versions of your application. You should validate the page format and print settings objects returned to you by calling the functions PMSessionValidatePageFormat and PMSessionValidatePrintSettings. Note that perfect translation between the old and new style objects is not achievable.

In Mac OS X, the function assumes the print record to be converted is a LaserWriter 8 print record.

Special Considerations

If you need to convert a Mac OS 9 print record into data you can use in Mac OS X, you should extract the page size data from the print record and use the function `PMCreatePageFormatWithPMPaper` (page 2144) to create a `PMPageFormat` object that corresponds to that data.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSessionCopyDestinationFormat

Obtains the destination format for a print job.

```
OSStatus PMSessionCopyDestinationFormat (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   CFStringRef *destFormatP
);
```

Parameters

printSession

The printing session that provides a context for the print job.

printSettings

The print settings object for the print job whose destination format you want to obtain.

destFormatP

A pointer to your `CFStringRef` variable. On return, the variable refers to a Core Foundation string that contains the destination format for the print job. You are responsible for releasing the string. Currently, there are two possible values: `kPMDocumentFormatPDF` or `kPMDocumentFormatPostScript`.

If an error occurs, the variable is set to `NULL`. If the function executes without error and the variable is set to `NULL`, the print job is set to use the default destination format.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`PMCore.h`

PMSessionCopyDestinationLocation

Obtains a destination location for a print job.

```
OSStatus PMSessionCopyDestinationLocation (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   CFURLRef *destLocationP
);
```

Parameters

printSession

The printing session that provides a context for the print job.

printSettings

The print settings for the print job whose destination location you want to obtain.

destLocationP

A pointer to your `CFURLRef` variable. On return, the variable refers to a Core Foundation URL that specifies the destination location of the print job. You are responsible for releasing the URL. If `NULL` is returned and the function executes without error (result code is `noErr`), the print job uses the default destination location for the current destination type. If an error occurs, the variable is set to `NULL`.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Some destination types define a specific kind of destination location for a print job. For example, the destination type `kPMDestinationFile` uses a file system URL to specify where a new file should be created for the print job's output.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`PMCore.h`

PMSessionCopyOutputFormatList

Obtains an array of destination formats supported by the current print destination.

```
OSStatus PMSessionCopyOutputFormatList (
   PMPrintSession printSession,
   PMDestinationType destType,
   CFArrayRef *documentFormatP
);
```

Parameters

printSession

The printing session that provides a context for the print job. The printer associated with this session is queried for the MIME types it supports.

destType

A destination type that specifies the destination for which you want to obtain valid destination formats. See "Destination Types" (page 2281) for a list of the possible destination types a print job can have.

documentFormatP

A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array that contains a list of destination formats that can be generated for the current print destination. See "Document Format Strings" (page 2282) for a list of some of the output formats that can be returned.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`PMCore.h`

PMSessionCreatePageFormatList

Obtains a list of page format objects, each of which describes a paper size available on the specified printer.

```
OSStatus PMSessionCreatePageFormatList (
    PMPrintSession printSession,
    PMPrinter printer,
    CFArrayRef *pageFormatList
);
```

Parameters

printSession

The current printing session.

printer

The printer whose list of page sizes you want to enumerate.

pageFormatList

A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array that contains the page format (`PMPageFormat`) objects associated with the specified printer. You are responsible for releasing the array. Each page format object describes a paper size available for the specified printer. If the function fails, then on return the array is `NULL`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

You can use this function to find the available sheet sizes (and the imageable area for them) for a given printer. After you obtain the page format list, you can call the function `PMGetUnadjustedPaperRect` (page 2171) for each page format object in the list to obtain the sheet rectangle size. Once you find the paper size you want, call `PMGetUnadjustedPageRect` (page 2170) to obtain the imageable area for that paper size.

Availability
Available in Mac OS X v10.1 and later.

Declared In
PMCore.h

PMSessionCreatePrinterList

Creates a list of printers available in the specified printing session.

```
OSStatus PMSessionCreatePrinterList (
   PMPrintSession printSession,
   CFArrayRef *printerList,
   CFIndex *currentIndex,
   PMPrinter *currentPrinter
);
```

Parameters

printSession
The printing session whose printer list you want to obtain.

printerList
A pointer to your CFArrayRef variable. On return, the variable refers to a Core Foundation array containing a list of printers available in the specified printing session. Each element in the array is a Core Foundation string that contains a printer's name as shown in the user interface. You are responsible for releasing the array.

currentIndex
A pointer to your CFIndex variable. On return, the variable contains a value specifying where the current printer is in the printer list.

currentPrinter
A pointer to your PMPrinter (page 2276) variable. On return, the variable refers to a printer object that represents the current printer. You should not release the printer object without first retaining it. If the printer is the generic printer, the variable is set to NULL.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function PMCreateSession (page 2145).

You can call the function PMSessionCreatePrinterList to obtain a valid printer name to pass to the function PMSessionSetCurrentPrinter.

Special Considerations

In Mac OS X v10.2 and later, Apple recommends using the function PMServerCreatePrinterList (page 2215) instead. PMServerCreatePrinterList doesn't require a PMSession object; it can be called at any time. It also works directly with PMPrinter objects.

Availability
Available in Mac OS X v10.1 and later.

See Also
PMServerCreatePrinterList (page 2215)

Declared In
PMCore.h

PMSessionDefaultPageFormat

Assigns default parameter values to a page format object used in the specified printing session.

```
OSStatus PMSessionDefaultPageFormat (
   PMPrintSession printSession,
   PMPageFormat pageFormat
);
```

Parameters

printSession

The printing session for the specified page format object.

pageFormat

The page format object to which you want to assign default values.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call the function `PMSessionDefaultPageFormat` between the creation and release of the printing session. See the function `PMCreateSession` (page 2145).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`PMCore.h`

PMSessionDefaultPrintSettings

Assigns default parameter values to a print settings object for the specified printing session.

```
OSStatus PMSessionDefaultPrintSettings (
   PMPrintSession printSession,
   PMPrintSettings printSettings
);
```

Parameters

printSession

The printing session for the specified print settings object.

printSettings

The print settings object to which you want to assign default values.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call the function `PMSessionDefaultPrintSettings` between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMSessionDisableColorSync

Disables use of a custom ColorSync profile previously enabled by the function `PMSessionEnableColorSync` (page 2226). (Deprecated in Mac OS X v10.5. There is no replacement; draw using Quartz 2D instead.)

```
OSStatus PMSessionDisableColorSync (
   PMPrintSession printSession
);
```

Parameters

printSession

The printing session whose page-specific ColorSync profile you want to disable.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call the `PMSessionDisableColorSync` function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must call this function within the scope of calls to the functions `PMSessionBeginPage` and `PMSessionEndPage`.

The function `PMSessionDisableColorSync` applies only to the current page. The function is useful only if the graphics context is QuickDraw and the current port is the printing port.

Special Considerations

This function is deprecated because QuickDraw is deprecated. When drawing with Quartz, the current stroke and fill color space and the color space associated with an image are used to characterize color. Quartz provides ways to use ColorSync profiles to create color spaces, so you can characterize color using ColorSync simply by drawing with Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionEnableColorSync

Enables use of a custom ColorSync profile previously set by the function `PMSetProfile` (page 2262). (Deprecated in Mac OS X v10.5. There is no replacement; draw using Quartz 2D instead.)

```
OSStatus PMSessionEnableColorSync (
   PMPrintSession printSession
);
```

Parameters

printSession

The printing session whose page-specific ColorSync profile you want to enable.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must call this function within the scope of calls to the functions `PMSessionBeginPage` and `PMSessionEndPage`.

The function `PMSessionEnableColorSync` applies only to the current page. The function is useful only if the graphics context is QuickDraw and the current port is the printing port.

Special Considerations

This function is deprecated because QuickDraw is deprecated. When drawing with Quartz, the current stroke and fill color space and the color space associated with an image are used to characterize color. Quartz provides ways to use ColorSync profiles to create color spaces, so you can characterize color using ColorSync simply by drawing with Quartz.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSessionEndDocumentNoDialog

Ends a print job started by calling the function `PMSessionBeginCGDocumentNoDialog` (page 2217) or `PMSessionBeginDocumentNoDialog` (page 2218).

```
OSStatus PMSessionEndDocumentNoDialog (
   PMPrintSession printSession
);
```

Parameters

printSession

The current printing session. On return, the printing session is no longer valid; however, you must still call the function `PMRelease` (page 2214) to release the object.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

This function is similar to the function `PMSessionEndDocument` except that the printing status dialog is suppressed.

This function is used to end a print job, and it should be called within your application's print loop after the call to the function `PMSessionEndPageNoDialog` and before releasing the printing session. The same printing session that is created by the function `PMCreateSession` for the Print dialog should be used for the print loop.

The function `PMSessionEndDocumentNoDialog` must be called after its corresponding `Begin` function (`PMSessionBeginCGDocumentNoDialog` (page 2217) or `PMSessionBeginDocumentNoDialog` (page 2218)). If the `Begin` function returns `noErr`, the function `PMSessionEndDocument` must be called, even if errors occur within the scope of the `Begin` and `End` functions. You should not call `PMSessionEndDocumentNoDialog` if the `Begin` function returns an error.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`PMCore.h`

PMSessionEndPageNoDialog

Indicates the end of drawing the current page for the specified printing session.

```
OSStatus PMSessionEndPageNoDialog (
   PMPrintSession printSession
);
```

Parameters

printSession
The printing session that provides a context for the print job.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is similar to the function `PMSessionEndPage` except that the printing status dialog is suppressed.

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must call the functions `PMSessionBeginPageNoDialog` and `PMSessionEndPageNoDialog` within the scope of calls to the `Begin` print job function (`PMSessionBeginCGDocumentNoDialog` (page 2217)) and the `End` print job function (`PMSessionEndDocumentNoDialog` (page 2227)).

If the function `PMSessionBeginPageNoDialog` returns `noErr`, you must later call the function `PMSessionEndPageNoDialog`, even if errors occur within the scope of `PMSessionBeginPageNoDialog` and `PMSessionEndPageNoDialog`. You should not call `PMSessionEndPageNoDialog` if `PMSessionBeginPageNoDialog` returns an error.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`PMCore.h`

PMSessionError

Obtains the result code for any error returned by the printing session.

```
OSStatus PMSessionError (
   PMPrintSession printSession
);
```

Parameters

printSession

The printing session whose last error you want to obtain.

Return Value

A result code. See "Core Printing Result Codes" (page 2298). The constant `kPMCancel` indicates the user canceled the current print job.

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

The `PMSessionError` function returns the last printing session error, not the last error from a printing function (`PMxxx`). Because most printing functions return a result code, the `PMSessionError` function is not required for general error checking. However, you can use `PMSessionError` in your print loop to determine if the user cancels the current print job or if any other errors occur during printing that are not explicitly returned by one of the other calls. For example, if the user clicks the Cancel button in the status dialog or presses Command-period on the keyboard, this function returns the constant `kPMCancel`. If this or any other error is encountered during the print loop, your application should call the appropriate functions (for example, `PMSessionEndPage` and `PMSessionEndDocument`) to exit the print loop before your application reports the error.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`PMCore.h`

PMSessionGeneral

Maintains compatibility with the `PrGeneral` function in the classic Printing Manager. (Deprecated in Mac OS X v10.4. Use `PMPrinterGetCommInfo` (page 2190) instead.)

```
OSStatus PMSessionGeneral (
   PMPrintSession printSession,
   Ptr pData
);
```

Parameters

printSession

The printing session whose data you want to obtain.

pData

A pointer to a `PrGeneral` data structure.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The function `PMSessionGeneral` is valid for the printing session passed to the function. In Mac OS X, the function `PMSessionGeneral` makes an attempt to get the requested data if the opcode is `getPSInfoOp`. Otherwise the result code `kPMNotImplemented` is returned.

For more information about using the function `PMSessionGeneral`, see *Supporting Printing in Your Carbon Application*.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionGetCGGraphicsContext

Obtains the Quartz graphics context for the current page in a printing session.

```
OSStatus PMSessionGetCGGraphicsContext (
   PMPrintSession printSession,
   CGContextRef *context
);
```

Parameters

printSession
The printing session whose Quartz graphics context you want to obtain.

context
A pointer to your `CGContextRef` (page 137) variable. On return, the variable refers to the Quartz graphics context for the current page in the specified printing session. The context's origin is at the lower-left corner of the sheet of paper, not the imageable area. You should not release the context without first retaining it. The context is valid only for the current page; you should not retain it beyond the end of the page.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
If you're using Quartz 2D to draw the content for a print job, after each call to `PMSessionBeginPage` you should call `PMSessionGetCGGraphicsContext` to obtain the Quartz graphics context for the current page. Note that before you can use the function `PMSessionGetCGGraphicsContext`, you must have called `PMSessionBeginCGDocument` or `PMSessionBeginCGDocumentNoDialog` (page 2217) instead of `PMSessionBeginDocument` or `PMSessionBeginDocumentNoDialog` (page 2218).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`PMCore.h`

PMSessionGetCurrentPrinter

Obtains the current printer associated with a printing session.

```
OSStatus PMSessionGetCurrentPrinter (
   PMPrintSession printSession,
   PMPrinter *currentPrinter
);
```

Parameters

printSession

The printing session whose printer you want to obtain.

currentPrinter

A pointer to your PMPrinter (page 2276) variable. On return, the variable refers to the printer associated with the specified printing session. The printer object is valid as long as the printing session is valid or the current printer hasn't changed. You should not release this object without first retaining it.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function PMCreateSession (page 2145).

Availability

Available in Mac OS X v10.0 and later.

See Also

PMSessionSetCurrentPMPrinter (page 2241)

Declared In

PMCore.h

PMSessionGetDataFromSession

Obtains application-specific data previously stored in a printing session object.

```
OSStatus PMSessionGetDataFromSession (
   PMPrintSession printSession,
   CFStringRef key,
   CFTypeRef *data
);
```

Parameters

printSession

The printing session whose data you want to obtain.

key

The key that uniquely identifies the data to be retrieved. You specify this key when you store the data using the function PMSessionSetDataInSession (page 2242).

data

A pointer to your CFTypeRef variable. On return, the variable refers to the data retrieved from the printing session.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMSessionSetDataInSession` (page 2242)

Declared In
`PMCore.h`

PMSessionGetDestinationType

Obtains the output destination for a print job.

```
OSStatus PMSessionGetDestinationType (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMDestinationType *destTypeP
);
```

Parameters

printSession

The printing session that provides a context for the print job. This must be the same printing session used for the Print dialog. The printing session contains the preview setting, which can override the destination type in the print settings.

printSettings

The print settings for the print job whose destination you want to obtain.

destTypeP

A pointer to your `PMDestinationType` variable. On return, the variable contains the destination type for the specified print job. Possible values include:

- `kPMDestinationPrinter` (output to a printer)
- `kPMDestinationFile` (output to a file)
- `kPMDestinationFax` (output to a fax)
- `kPMDestinationPreview` (output to print preview)
- `kPMDestinationProcessPDF` (output to a PDF workflow option)

See "Destination Types" (page 2281) for a complete description of the destination type constants.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

All of the destination types are stored in the print settings object except for `kPMDestinationPreview`, which is stored in the printing session object. If the destination type is set as preview, the preview setting overrides the destination set in the print settings object.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`PMCore.h`

PMSessionGetDocumentFormatGeneration

Obtains the spool file formats that can be generated for the specified printing session. (Deprecated in Mac OS X v10.4. If you're drawing using Quartz 2D instead of QuickDraw, use `PMSessionBeginCGDocument` or `PMSessionBeginCGDocumentNoDialog` (page 2217); for submitting PostScript data, use `PMPrinterPrintWithFile` (page 2203) or `PMPrinterPrintWithProvider` (page 2204); to draw EPS data, use `PMCGImageCreateWithEPSDataProvider` (page 2139).)

```
OSStatus PMSessionGetDocumentFormatGeneration (
   PMPrintSession printSession,
   CFArrayRef *docFormats
);
```

Parameters

`printSession`
The printing session whose spool file formats you want to obtain.

`docFormats`
A pointer to your `CFArrayRef` variable. On return, the variable refers to a Core Foundation array that contains the MIME types for the available spool file formats. Each element in the array is a Core Foundation string. Despite what its name implies, the function `PMSessionGetDocumentFormatGeneration` has Create/Copy semantics which means you are responsible for releasing the array.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call the function `PMSessionGetDocumentFormatGeneration` between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You should call `PMSessionGetDocumentFormatGeneration` only after the Print dialog is dismissed.

The function `PMSessionGetDocumentFormatGeneration` determines the spool file formats that the specific print job supports. Spool file formats are represented by MIME types. The Mac OS X print spooler supports PDF and PICT + PS. The default spool file format is PDF. PICT + PS is supported only for printing to a PostScript printer.

Special Considerations

The PICT + PS spool file format is not available on Intel-based systems.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

See Also
PMSessionSetDocumentFormatGeneration (page 2244)

Declared In
PMCoreDeprecated.h

PMSessionGetDocumentFormatSupported

Obtains the spool file formats that are accepted by the current printer driver. (Deprecated in Mac OS X v10.4. Use PMPrinterPrintWithFile (page 2203), PMPrinterPrintWithProvider (page 2204), or PMCGImageCreateWithEPSDataProvider (page 2139) instead.)

```
OSStatus PMSessionGetDocumentFormatSupported (
   PMPrintSession printSession,
   CFArrayRef *docFormats,
   UInt32 limit
);
```

Parameters

printSession
The current printing session.

docFormats
On return, an array of CFString values containing MIME types specifying the spool file formats supported by the current printer driver. See "Document Format Strings" for a description of possible return values.

limit
The maximum number of supported document formats to be returned.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid within the context of a printing session.

Spool file formats are represented by MIME types. In Mac OS X, printer modules may support a wide range of spool file formats. The first item in the list of supported spool file formats is the default for the current printer driver.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
PMCoreDeprecated.h

PMSessionGetGraphicsContext

Obtains the graphics context for the current page in a printing session. (Deprecated in Mac OS X v10.5. Use PMSessionGetCGGraphicsContext (page 2230) instead.)

```
OSStatus PMSessionGetGraphicsContext (
   PMPrintSession printSession,
   CFStringRef graphicsContextType,
   void **graphicsContext
);
```

Parameters

printSession

The printing session whose current graphics context you want to obtain.

graphicsType

The desired graphics context type. This parameter is currently ignored.

graphicsContext

On return, a reference to the current graphics context. The graphics context returned is the one last set by a call to the function `PMSessionSetDocumentFormatGeneration` or the default (QuickDraw) if there was no call to the function. You must typecast the context to an appropriate graphics type, either `grafPtr` or `CGContextRef` (page 137).

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must also call the function `PMSessionGetGraphicsContext` within the scope of the functions `PMSessionBeginPage` and `PMSessionEndPage`.

In Mac OS X v10.3 and earlier, you should call this function for each page you draw for a print job. After each call to the function `PMSessionBeginPage` your application should call `PMSessionGetGraphicsContext` to obtain the current graphics context. If that context is a QuickDraw context, then set the drawing port to this port by calling the QuickDraw `SetPort` function. See the discussion of the function `PMSessionBeginPage` for more information.

Special Considerations

In Mac OS X v10.4 and later, Apple recommends using the function `PMSessionGetCGGraphicsContext` (page 2230) instead of this function. QuickDraw is deprecated and your application should be using Quartz 2D for its rendering.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

Declared In

`PMCoreDeprecated.h`

PMSessionIsDocumentFormatSupported

Reports whether the current printer driver supports a specified spool file format. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMSessionIsDocumentFormatSupported (
   PMPrintSession printSession,
   CFStringRef docFormat,
   Boolean *supported
);
```

Parameters

printSession

The current printing session.

docFormat

A spool file format represented by a MIME type.

supported

Returns `true` if the spool file format is supported by the current printer driver.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Valid within the context of a printing session.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSessionMakeOldPrintRecord

Creates an old-style print record from page format and print settings objects. (Deprecated in Mac OS X v10.4. There is no replacement; old-style print records are obsolete and serve no useful purpose in Mac OS X.)

```
OSStatus PMSessionMakeOldPrintRecord (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat pageFormat,
   Handle *printRecordHandle
);
```

Parameters

printSession

The current printing session.

printSettings

A print settings object. To create a print settings object you can call the function `PMCreatePrintSettings` (page 2144) and then call the function `PMSessionDefaultPrintSettings` (page 2225) to initialize the print settings object to default values.

pageFormat

A page format object. To create a page format object you can call the function `PMCreatePageFormat` (page 2143) and then call the function `PMSessionDefaultPageFormat` (page 2225) to initialize the page format object to default values.

printRecordHandle
On return, a handle to an old-style print record. You are responsible for disposing of the handle.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

You can use `PMSessionMakeOldPrintRecord` to create an old-style print record to store with your documents for compatibility with pre-Carbon versions of your application. Note that because the page format and print settings objects contain more information than the old print record, some settings may be lost in the conversion. That is, perfect translation between the old and new style objects is not achievable.

In Mac OS X, the function always creates a LaserWriter 8 compatible print record.

Special Considerations

The proper way to keep page format information for use in Mac OS X is with a flattened `PMPageFormat` object. Typically applications don't keep print settings with a document but if that is appropriate for a given application, the proper way to do so is to use a flattened `PMPrintSettings` object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionPostScriptBegin

Puts the current printer driver into PostScript mode, ready to accept PostScript data instead of QuickDraw data. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMSessionPostScriptBegin (
   PMPrintSession printSession
);
```

Parameters

printSession
The current printing session.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call the function `PMSessionPostScriptBegin` between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must also call the function within the scope of the functions `PMSessionBeginPage` and `PMSessionEndPage`.

To ensure that the current printer driver supports PostScript data, call `PMSessionGetDocumentFormatGeneration` before you call the function `PMSessionPostScriptBegin`. Check the list of supported spool file formats. If PICT + PS is one of them, select that format by calling the function `PMSessionSetDocumentFormatGeneration`. The function `PMSessionSetDocumentFormatGeneration` must be called before you call `PMSessionBeginDocument`.

The function `PMSessionPostScriptBegin` is not useful unless the current port is the printing port. The function returns `true` if the document format is not PICT + PS.

Special Considerations

The PICT + PS spool file format is not available on Intel-based systems.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSessionPostScriptData

Passes PostScript data, referenced by a pointer, to the current printer driver. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMSessionPostScriptData (
   PMPrintSession printSession,
   Ptr psPtr,
   Size len
);
```

Parameters

printSession

The current printing session.

psPtr

A pointer to the PostScript data you want to pass to the current printer driver.

len

The number of bytes of PostScript data.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). Typically you call this function with the scope of calls to the functions `PMSessionPostScriptBegin` and `PMSessionPostScriptEnd`.

The function `PMSessionPostScriptData` is not useful unless the current port is the printing port and the document format is PICT + PS.

Special Considerations

The PICT + PS spool file format is not available on Intel-based systems.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionPostScriptEnd

Restores the current driver to QuickDraw mode, ready to accept QuickDraw data instead of PostScript data. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMSessionPostScriptEnd (
   PMPrintSession printSession
);
```

Parameters

printSession
The current printing session.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must also call this function with the scope of calls to the functions `PMSessionBeginPage` and `PMSessionEndPage`.

You call the function `PMSessionPostScriptEnd` to complete a PostScript block started with `PMSessionPostScriptBegin`. The function `PMSessionPostScriptEnd` is not useful unless the current port is the printing port and the document format is PICT + PS.

Special Considerations

The PICT + PS spool file format is not available on Intel-based systems.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionPostScriptFile

Passes the PostScript data, contained in a file, to the current printer driver. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMSessionPostScriptFile (
   PMPrintSession printSession,
   FSSpec *psFile
);
```

Parameters

printSession

The current printing session.

psFile

A pointer to a variable that specifies a file location. The file should contain the PostScript data you want to pass to the current printer driver.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You typically call this function within the scope of calls to the function `PMSessionPostScriptBegin` and `PMSessionPostScriptEnd`.

The function `PMSessionPostScriptFile` is not useful unless the current port is the printing port and the document format is PICT + PS.

Special Considerations

The PICT + PS spool file format is not available on Intel-based systems.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSessionPostScriptHandle

Passes the PostScript data, referenced by a Memory Manager handle, to the current printer driver. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203), `PMPrinterPrintWithProvider` (page 2204), or `PMCGImageCreateWithEPSDataProvider` (page 2139) instead.)

```
OSStatus PMSessionPostScriptHandle (
   PMPrintSession printSession,
   Handle psHandle
);
```

Parameters

printSession

The current printing session.

psHandle

A handle to the PostScript data you want to pass to the current printer driver. You must make sure the handle is of the appropriate size for the data, otherwise you risk corrupting the spool file.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must also call this function within the scope of calls to the function `PMSessionPostScriptBegin` and `PMSessionPostScriptEnd`.

The function `PMSessionPostScriptEnd` is not useful unless the current port is the printing port and the document format is PICT + PS.

Special Considerations

The PICT + PS spool file format is not available on Intel-based systems.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionSetCurrentPMPrinter

Changes the current printer for a printing session.

```
OSStatus PMSessionSetCurrentPMPrinter (
   PMPrintSession session,
   PMPrinter printer
);
```

Parameters

session
The printing session whose printer you want to change.

printer
The new printer for the printing session.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability
Available in Mac OS X v10.3 and later.

See Also
`PMSessionGetCurrentPrinter` (page 2231)

Declared In
`PMCore.h`

PMSessionSetCurrentPrinter

Changes the current printer for a printing session to a printer specified by name. (Deprecated in Mac OS X v10.4. Use `PMSessionSetCurrentPMPrinter` (page 2241) instead.)

```
OSStatus PMSessionSetCurrentPrinter (
   PMPrintSession session,
   CFStringRef printerName
);
```

Parameters

session

The printing session whose printer you want to change.

printerName

The name of the printer you want to set as the current printer.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSessionSetDataInSession

Stores your application-specific data in a printing session object.

```
OSStatus PMSessionSetDataInSession (
   PMPrintSession printSession,
   CFStringRef key,
   CFTypeRef data
);
```

Parameters

printSession

The printing session in which you want to store application-specific data.

key

A key that uniquely identifies the data being added. This key is required to retrieve the data using the function `PMSessionGetDataFromSession` (page 2231).

data

The data to be stored in the printing session.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMSessionGetDataFromSession` (page 2231)

Declared In
`PMCore.h`

PMSessionSetDestination

Sets the destination location, format, and type for a print job.

```
OSStatus PMSessionSetDestination (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMDestinationType destType,
   CFStringRef destFormat,
   CFURLRef destLocation
);
```

Parameters

printSession
The printing session that provides a context for the print job.

printSettings
The print settings for the print job whose destination you want to set.

destType
The destination type for the print job associated with the specified printing session and print settings. Possible values include:

- `kPMDestinationPrinter` (output to a printer)
- `kPMDestinationFile` (output to a file)
- `kPMDestinationFax` (output to a fax)
- `kPMDestinationPreview` (output to print preview)
- `kPMDestinationProcessPDF` (output to a PDF workflow option)

See "Destination Types" (page 2281) for a complete description of destination types you can specify.

destFormat
The MIME type to be generated for the specified destination type. Pass `NULL` if you want to use the default format for the specified destination type. To obtain a list of valid formats for a given destination type, use the function `PMSessionCopyOutputFormatList` (page 2222).

destLocation
A reference to a Core Foundation URL that specifies a destination location. You can provide this if the destination type supports a destination location. Otherwise, pass `NULL`. For example, if the destination type is a file (`kPMDestinationFile`) you can supply a file system URL to specify where the file resides.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can use the function `PMSessionSetDestination` when you want to send print output to a file without requiring user interaction. You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`PMCore.h`

PMSessionSetDocumentFormatGeneration

Requests a specified spool file format and supplies the graphics context type to use for drawing pages within the print loop. (Deprecated in Mac OS X v10.4. If you're drawing using Quartz 2D instead of QuickDraw, use `PMSessionBeginCGDocument` or `PMSessionBeginCGDocumentNoDialog` (page 2217); for submitting PostScript data, use `PMPrinterPrintWithFile` (page 2203) or `PMPrinterPrintWithProvider` (page 2204); to draw EPS data, use `PMCGImageCreateWithEPSDataProvider` (page 2139).)

```
OSStatus PMSessionSetDocumentFormatGeneration (
   PMPrintSession printSession,
   CFStringRef docFormat,
   CFArrayRef graphicsContextTypes,
   CFTypeRef options
);
```

Parameters

printSession
The printing session whose spool file format and graphics context type you want to specify.

docFormat
A Core Foundation string that specifies the desired spool file format as a MIME type. See "Document Format Strings" (page 2282) for a description of the constants you can use to specify the document format.

graphicsContexts
A reference to a Core Foundation array of graphics contexts to use for drawing pages within the print loop. You can supply a QuickDraw graphics context (`kPMGraphicsContextQuickDraw`) or a Quartz 2D graphics context (`kPMGraphicsContextCoreGraphics`). An array of length 1 is the only length that is supported, regardless of graphics context type. See "Graphics Context Types" (page 2283) for a description of the constants you can use to specify a graphics context.

options
Reserved for future use.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You only need to call the function `PMSessionSetDocumentFormatGeneration` if you want to specify a format other than the default format (PDF) or a graphics context other than the default context (QuickDraw). If you want to use the default format for the operating system and to draw with QuickDraw, then you do not

need to call this function. If you want to generate PICT + PS to use as one of the supported formats, then call `PMSessionSetDocumentFormatGeneration` to set the graphics context to QuickDraw and the format to PICT + PS. Note that the PICT + PS format is not available on Intel-based systems.

If you want to use a Quartz 2D graphics context to draw each page, you can call the following code to inform the printing system in all versions of Mac OS X.

```
static OSStatus MyPMSessionBeginCGDocument (
    PMPrintSession printSession,
    PMPrintSettings printSettings,
    PMPageFormat pageFormat)
{
    OSStatus err = noErr;

    // Use the simpler call if it is present.
    if(&PMSessionBeginCGDocument != NULL) {
       err = PMSessionBeginCGDocument (printSession, printSettings, pageFormat);
    }
    else {
        CFStringRef s[1] = { kPMGraphicsContextCoreGraphics };
        CFArrayRef graphicsContextsArray = CFArrayCreate (
            kCFAllocatorDefault, (const void**)s, 1, &kCFTypeArrayCallBacks);
        err = PMSessionSetDocumentFormatGeneration (
            printSession, kPMDocumentFormatPDF, graphicsContextsArray, NULL);
        CFRelease (graphicsContextsArray);
        if(!err)
            err = PMSessionBeginDocument (
                printSession, printSettings, pageFormat);
    }
    return err;
}
```

The previous code informs the printing system that you want a Quartz graphics context, but you get the actual context for your printing port only after you call the function `PMSessionBeginPage` and then call the following code.

```
static OSStatus MyPMSessionGetCGGraphicsContext (
    PMPrintSession printSession,
    CGContextRef *printingContextP)
{
    OSStatus err = noErr;

    // Use the simpler call if it is present.
    if(&PMSessionGetCGGraphicsContext != NULL) {
        err = PMSessionGetCGGraphicsContext (printSession, printingContextP);
    }
    else {
        err = PMSessionGetGraphicsContext (
            printSession, kPMGraphicsContextCoreGraphics,
            (void**)printingContextP);
    }
    return err;
}
```

The printing context you get is a Quartz context into which you can draw. Note that the default coordinate system for Quartz 2D is not the same as that used for QuickDraw. Quartz 2D defines the coordinates of the lower-left corner of the sheet as (0,0) whereas the origin for the QuickDraw port is the upper-left corner of the imageable area.

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You must call the function `PMSessionSetDocumentFormatGeneration` before you call `PMSessionBeginDocument` or `PMSessionBeginDocumentNoDialog` (page 2218). Before requesting a spool file format using this function, you should call the function `PMSessionGetDocumentFormatGeneration` to get the list of supported formats.

Special Considerations

The PICT + PS spool file format is not available on Intel-based systems.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

See Also

`PMSessionGetDocumentFormatGeneration` (page 2233)

Related Sample Code

CarbonSketch

Declared In

`PMCoreDeprecated.h`

PMSessionSetError

Sets the value of the current result code for the specified printing session.

```
OSStatus PMSessionSetError (
   PMPrintSession printSession,
   OSStatus printError
);
```

Parameters

printSession

The printing session whose result code you want to set.

printError

The result code you want to set. This result code is returned by the `PMSessionError` function.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

You can use this function to terminate a printing session if your application encounters any errors inside the print loop. Typically, this function is used by an application's idle function. The idle function isn't called in Mac OS X, so this usage is not available.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMCore.h`

PMSessionSetIdleProc

Installs an idle callback function in your print loop. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
OSStatus PMSessionSetIdleProc (
   PMPrintSession printSession,
   PMIdleUPP idleProc
);
```

Parameters

printSession

The printing session that provides a context for the print job.

idleProc

A universal procedure pointer to your idle function. Your idle function is defined by the callback `PMIdleProcPtr` (page 2273).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You do not need this function in Mac OS X. Instead, use the standard idle proc.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSessionSetPSInjectionData

Specifies a set of PostScript code injection points and the PostScript data to be injected. (Deprecated in Mac OS X v10.4. Use `PMPrinterPrintWithFile` (page 2203) or `PMPrinterPrintWithProvider` (page 2204) instead.)

```
OSStatus PMSessionSetPSInjectionData (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   CFArrayRef injectionDictArray
);
```

Parameters

printSession

The current printing session.

printSettings

The print settings object in which to place the specified injection points.

injectionDictArray

A reference to a Core Foundation array that contains one or more Core Foundation dictionary (`CFDictionary`) entries. Each dictionary entry specifies PostScript injection data you want inserted at a specific point in the print stream. See "PostScript Injection Dictionary Keys" (page 2285) for a description of the constants you can use as keys for these dictionary entries.

Return Value

A result code. See "Core Printing Result Codes" (page 2298). The result code `kPMInvalidParameter` is returned if the `injectionDictArray` object contains any invalid entries. The result code `kPMInvalidPrintSession` is returned if the document format has not been set to `kPMDocumentFormatPICTPS` for the specified printing session.

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). Before calling `PMSessionSetPSInjectionData`, your application must set the document format of the printing session to `kPMDocumentFormatPICTPS` using the function `PMSessionSetDocumentFormatGeneration` (page 2244).

For applications that require extensive control over PostScript code generation, the function `PMSessionSetPSInjectionData` provides the ability to insert PostScript code into specified places in the print stream. It is intended for use by desktop publishing applications for which functions such as `PMSessionPostScriptData` do not provide sufficient control.

You specify the injection points by creating an array of `CFDictionary` entries. Each dictionary consists of key-value pairs in which the key specifies where to inject the PostScript and the value specifies the PostScript data you want to inject. The function verifies that the `injectionDictArray` parameter is properly formed, and returns the `kPMInvalidParameter` result code if the array does not contain valid entries.

If you call the function `PMSessionSetPSInjectionData` a second time for a given print settings object, the injection points you specified previously are replaced with the new injection points.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSessionValidatePageFormat

Updates the values in a page format object and validates them against the current formatting printer.

```
OSStatus PMSessionValidatePageFormat (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   Boolean *result
);
```

Parameters

printSession

The printing session for the specified page format object.

pageFormat
The page format object to validate.

result
A pointer to your Boolean variable. On return, `true` if the function set the page format object to default values; otherwise, `false`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of the printing session. See the function `PMCreateSession` (page 2145).

The function `PMSessionValidatePageFormat` validates the page format object against the current formatting printer. The formatting printer is displayed in the Format for pop-up menu in the Page Setup dialog. The default formatting printer is the generic Any Printer. If the page format object contains values that are not valid for the formatting printer, the page format object is set to default values and the *result* parameter is set to `true`.

Validating a page format object also causes calculated fields (such as the adjusted paper and page rectangles) to be updated based on the changed settings (such as resolution, scaling, and page orientation). If the page format object contains values that are valid for the formatting printer but need to be updated, the *result* parameter is set to `false`.

After you call any function that makes changes to a page format object (such as `PMSetOrientation`), you should call the function `PMSessionValidatePageFormat` to validate the page format object before using that object.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMSessionValidatePrintSettings

Validates a print settings object within the context of the specified printing session.

```
OSStatus PMSessionValidatePrintSettings (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   Boolean *result
);
```

Parameters
printSession
The printing session for the specified print settings object.

printSettings
The print settings object to validate.

result
A pointer to your Boolean variable. On return, `true` if any parameters changed, or `false` if no parameters changed.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMSetAdjustedPageRect

Requests a particular page size, adjusted for the current rotation, resolution, or scaling settings. (Deprecated in Mac OS X v10.5. To set a particular paper size and margins, obtain or create a `PMPaper` (page 2275) object and call `PMCreatePageFormatWithPMPaper` (page 2144).)

```
OSStatus PMSetAdjustedPageRect (
   PMPageFormat pageFormat,
   const PMRect *pageRect
);
```

Parameters

pageFormat
The page format object whose page rectangle you want to set.

pageRect
A pointer to your `PMRect` (page 2277) data structure that specifies the desired size of the page rectangle, in points. The top-left coordinates should be (0,0). See *Supporting Printing in Your Carbon Application* for more information on page and paper rectangles.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
This function is not recommended. You should call this function only if your application provides desktop publishing and the Page Setup dialog does not provide sufficient control. Typically, such applications display their own specialized document format dialog.

If you decide to use this function, you must call the function between the creation and release of a printing session. See the function `PMCreateSession` (page 2145). You can use `PMSetAdjustedPageRect` to set a drawing rectangle without going through the Page Setup dialog or calling other page format accessor functions. This function allows an application to specify the dimensions of the imageable area into which it draws.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSetCollate

Specifies whether the job collate option is selected.

```
OSStatus PMSetCollate (
   PMPrintSettings printSettings,
   Boolean collate
);
```

Parameters

`printSettings`
The print settings object whose job collate option you want to set.

`collate`
If `true`, the job collate option is selected; if `false` the option is not selected.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The Collated checkbox is displayed in the Copies & Pages pane of the Print dialog. This option determines how printed material is organized. For example, if you have a document that is three pages long and you are printing multiple copies with the Collated option selected, the job prints pages 1, 2, and 3 in that order and then repeats. However, if the Collated option is not selected and you're printing multiple copies of those same three pages, the job prints copies of page 1, then copies of page 2, and finally copies of page 3.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.2 and later.

See Also
`PMGetCollate` (page 2154)

Declared In
`PMCore.h`

PMSetColorMode

Sets the desired color mode for the print job. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
OSStatus PMSetColorMode (
   PMPrintSettings printSettings,
   PMColorMode colorMode
);
```

Parameters

printSettings

The print settings object whose color mode you want to set.

colorMode

The desired color mode. See "Color Modes" (page 2298) for a list of possible values.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Special Considerations

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMCoreDeprecated.h`

PMSetCopies

Sets the initial value for the number of copies to be printed.

```
OSStatus PMSetCopies (
   PMPrintSettings printSettings,
   UInt32 copies,
   Boolean lock
);
```

Parameters

printSettings

The print settings object you want to initialize.

copies

The initial value of the number of copies to print.

lock

The lock state of the setting. Locking is not supported at this time.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

If you call this function after initiating a print job, the change is ignored for the current job.

Availability

Available in Mac OS X v10.0 and later.

See Also

`PMGetCopies` (page 2155)

Declared In
PMCore.h

PMSetDuplex

Sets the duplex mode.

```
OSStatus PMSetDuplex (
   PMPrintSettings printSettings,
   PMDuplexMode duplexSetting
);
```

Parameters

printSettings

The print settings object whose duplex mode you want to set.

duplexSetting

The new duplex mode setting. Possible values include:

- kPMDuplexNone (one-sided printing)
- kPMDuplexNoTumble (two-sided printing)
- kPMDuplexTumble (two-sided printing with tumbling)

See "Duplex Modes" (page 2282) for a full description of the constants you can use to specify the new setting.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Duplex printing is a print job that prints on both sides of the paper. Two-Sided printing controls are displayed in the Layout pane of the Print dialog. Note that not all printers support duplex printing. This function specifies a setting that might not be available on a given destination.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.4 and later.

See Also
PMGetDuplex (page 2158)

Declared In
PMCore.h

PMSetError

Sets the value of the current result code. (Deprecated in Mac OS X v10.4. Use PMSessionSetError (page 2246) instead.)

```
OSStatus PMSetError (
    OSStatus printError
);
```

Parameters

printError

The result code you wish to set. This result code will be returned by the `PMError` function.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling `PMBegin`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMCoreDeprecated.h`

PMSetFirstPage

Sets the default page number of the first page to be printed.

```
OSStatus PMSetFirstPage (
   PMPrintSettings printSettings,
   UInt32 first,
   Boolean lock
);
```

Parameters

printSettings

The print settings object whose first page number you want to set.

first

The page number of the first page to print. This value appears in the From field of the Print dialog.

lock

The lock state of the setting. Locking is not supported at this time.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Typically, this function isn't used. In Mac OS X, if you call the function `PMSetPageRange` (page 2259) and then call `PMSetFirstPage` or `PMSetLastPage` using the same page range you specified for `PMSetPageRange`, then the Print dialog shows the From button selected. If you use the constant `kPMPrintAllPages` to set the page range with the function `PMSetPageRange`, then the Print dialog opens with the All button selected regardless of whether you also call `PMSetFirstPage` or `PMSetLastPage`.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.0 and later.

See Also
PMGetFirstPage (page 2159)

Related Sample Code
CarbonSketch

Declared In
PMCore.h

PMSetIdleProc

Installs an idle callback function in your print loop. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
OSStatus PMSetIdleProc (
    PMIdleUPP idleProc
);
```

Parameters

idleProc

A universal procedure pointer to your idle function. Your idle function is defined by the callback PMIdleProcPtr.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling PMBegin. The printing system calls your idle function periodically during your print loop.

Special Considerations

Your idle function is not called in Mac OS X. It's only called in Mac OS 8 and 9.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMSetJobName

Specifies the name of the print job. (Deprecated in Mac OS X v10.4. Use PMPrintSettingsSetJobName (page 2211) instead.)

```
OSStatus PMSetJobName (
   PMPrintSettings printSettings,
   StringPtr name
);
```

Parameters

printSettings

A PMPrintSettings object.

name

The name to assign to the print job. This string will be used to name the spool file.

Return Value
A result code. The result code `kPMInvalidParameter` is returned if you attempt to set the job name to an invalid file name or a null string.

Discussion
Valid after calling `PMBegin` and creating a print settings object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSetJobNameCFString

Specifies the name of a print job. (Deprecated in Mac OS X v10.5. Use `PMPrintSettingsSetJobName` (page 2211) instead.)

```
OSStatus PMSetJobNameCFString (
   PMPrintSettings printSettings,
   CFStringRef name
);
```

Parameters

printSettings

The print settings object whose job name you want to set.

name

The new name for the print job.

Return Value
A result code. See "Core Printing Result Codes" (page 2298). The result code `kPMInvalidParameter` is returned if you pass `NULL` or an empty string in the `name` parameter.

Discussion
You should call this function before you open the Print dialog.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

See Also
`PMGetJobNameCFString` (page 2161)

Related Sample Code
CarbonSketch

Declared In
PMCoreDeprecated.h

PMSetLastPage

Sets the page number of the last page to be printed.

```
OSStatus PMSetLastPage (
   PMPrintSettings printSettings,
   UInt32 last,
   Boolean lock
);
```

Parameters

printSettings

The print settings object whose last page number you want to set.

last

The page number of the last page to print. This value appears in the To field of the Print dialog. Pass the constant kPMPrintAllPages to print the entire document.

lock

The lock state of the setting. Locking is not supported at this time.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Typically, you call this function after the Print dialog is displayed to indicate the number of the last page number to be printed. In Mac OS X, setting the last page provides information used by the progress dialog that is shown during printing.

If you call the function PMSetPageRange (page 2259) and then call PMSetFirstPage or PMSetLastPage using the same page range you specified for PMSetPageRange, then the Print dialog shows the From button selected. If you use the constant kPMPrintAllPages to set the page range with the function PMSetPageRange, then the Print dialog opens with the All button selected regardless of whether you also call PMSetFirstPage or PMSetLastPage.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.0 and later.

See Also
PMGetLastPage (page 2162)

Related Sample Code
CarbonSketch

Declared In
PMCore.h

PMSetOrientation

Sets the page orientation for printing.

```
OSStatus PMSetOrientation (
   PMPageFormat pageFormat,
   PMOrientation orientation,
   Boolean lock
);
```

Parameters

pageFormat

The page format object whose page orientation you want to set.

orientation

A constant specifying the desired page orientation. Supported values are:

- `kPMPortrait`
- `kPMLandscape`
- `kPMReversePortrait` (Mac OS X v10.5 and later)
- `kPMReverseLandscape`

See "Page Orientation Constants" (page 2283) for a full description of the values you can use to specify page orientation.

lock

The lock state of the setting. You should pass `kPMUnlocked`. Locking is not supported at this time.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Special Considerations

In Mac OS X 10.4 and earlier, if you want to set the page orientation you need to call this function before initiating the print job (for example, by calling `PMSessionBeginCGDocument`). The page orientation you set applies to the entire print job. In Mac OS X 10.5 and later, you can use this function to change the orientation of an individual page in a print job by passing the updated page format to `PMSessionBeginPage` or `PMSessionBeginPageNoDialog`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`PMGetOrientation` (page 2163)

Declared In

`PMCore.h`

PMSetPageFormatExtendedData

Stores your application-specific data in a page format object.

```
OSStatus PMSetPageFormatExtendedData (
   PMPageFormat pageFormat,
   OSType dataID,
   UInt32 size,
   void *extendedData
);
```

Parameters

pageFormat

The page format object in which to store your extended data.

dataID

A 4-character code that identifies your data. This is typically your application's creator code. If your creator code is outside the ASCII 7-bit character range 0x20–0x7F, you need to use a different 4-character code.

size

The size, in bytes, of the data to be stored in the page format object.

extendedData

A pointer to the application-specific data you want to store in the page format object.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can retrieve the data you store with the function `PMSetPageFormatExtendedData` by calling the function `PMGetPageFormatExtendedData`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMGetPageFormatExtendedData` (page 2164)

Declared In
`PMCore.h`

PMSetPageRange

Sets the valid range of pages that can be printed.

```
OSStatus PMSetPageRange (
   PMPrintSettings printSettings,
   UInt32 minPage,
   UInt32 maxPage
);
```

Parameters

printSettings

The print settings object whose page range you want to set.

minPage

The minimum page number allowed. This value appears as the default in the From field of the Print dialog.

maxPage

The maximum page number allowed. This value appears as the default in the To field of the Print dialog. Pass the constant `kPMPrintAllPages` to allow the user to print the entire document. If the first page is set to 1, then passing `kPMPrintAllPages` as the maximum page number causes the All button to be selected.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
The function `PMSetPageRange` allows applications to set the minimum and maximum page numbers that can be printed for a document. If the user enters a value outside of this range in the Print dialog, the value is set to the closest allowed value. You can use the `PMGetFirstPage` (page 2159) and `PMGetLastPage` (page 2162) functions to obtain the values entered by the user in the Print dialog.)

If you call the function `PMSetPageRange` to set the maximum page to a value other than the constant `kPMPrintAllPages`, the function `PMSetPageRange` causes the page range in the Print dialog to be properly restricted to the specified range. If you call the function `PMSetPageRange` without also calling the functions `PMSetFirstPage` or `PMSetLastPage`, then the Print dialog shows the specified page range in the From and To fields but with the All button selected. If you call the function `PMSetPageRange` and then call `PMSetFirstPage` or `PMSetLastPage` using the same page range you specified for `PMSetPageRange`, then the Print dialog shows the From button selected.

In all cases, if your application sets a range with `PMSetPageRange` and subsequently calls `PMSetFirstPage` (page 2254) or `PMSetLastPage` (page 2257) with values outside of the specified range, Core Printing returns a result code of `kPMValueOutOfRange`. Conversely, if your application calls `PMSetPageRange` after calling `PMSetFirstPage` or `PMSetLastPage` (or after displaying the Print dialog), the page range specified by `PMSetPageRange` takes precedence, and the first and last page values are adjusted accordingly.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMGetPageRange` (page 2165)

Related Sample Code
CarbonSketch

Declared In
`PMCore.h`

PMSetPhysicalPaperSize

Requests a particular paper size, unaffected by rotation, resolution, or scaling. (Deprecated in Mac OS X v10.4. Use `PMCreatePageFormatWithPMPaper` (page 2144) instead.)

```
OSStatus PMSetPhysicalPaperSize (
   PMPageFormat pageFormat,
   const PMRect *paperSize
);
```

Parameters

pageFormat

The `PMPageFormat` which will hold the new physical paper size.

paperSize

The desired paper size expressed as a `PMRect`. The units are 1/72 inch. A `PMRect` is a rectangle whose individual components are doubles.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSetPrintSettingsExtendedData

Stores your application-specific data in a print settings object.

```
OSStatus PMSetPrintSettingsExtendedData (
   PMPrintSettings printSettings,
   OSType dataID,
   UInt32 size,
   void *extendedData
);
```

Parameters

printSettings

The print settings object in which to store your application-specific data.

dataID

A 4-character code that will be used to identify your data. The 4-character code must not contain any characters outside the standard ASCII 7-bit character range 0x20–0x7F. This is typically your application's creator code.

size

The size, in bytes, of the data to be stored in the print settings object.

extendedData

A pointer to a buffer that contains the extended data you want to store.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can retrieve the data you store with the function `PMSetPrintSettingsExtendedData` by calling the function `PMGetPrintSettingsExtendedData`.

You may find it easier to use the functions `PMPrintSettingsSetValue` (page 2212) and `PMPrintSettingsGetValue` (page 2210) to store and retrieve user-defined data in a print settings object. If you use these functions, make sure that the custom keys you define for your private data do not conflict with other print settings keys.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMGetPrintSettingsExtendedData` (page 2168)

Declared In
`PMCore.h`

PMSetProfile

Embeds a color profile during printing. (Deprecated in Mac OS X v10.5. There is no replacement; draw using Quartz 2D instead.)

```
OSStatus PMSetProfile (
   PMPrintSettings printSettings,
   PMTag tag,
   const CMProfileLocation *profile
);
```

Parameters

printSettings

The print settings object in which to embed the color profile.

tag

A tag that describes the usage of the profile. Currently, the only tag value you can pass is the constant `kPMSourceProfile`. See "Tag Constants" (page 2295) for more information on this constant.

profile

A pointer to a structure of type `CMProfileLocation` that specifies the location of a ColorSync profile. The profile must be version 2 or later. If you pass a profile that is an earlier version, the function returns the result code `kPMNotImplemented`.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can use the function `PMSetProfile` to tag QuickDraw drawing with a custom ColorSync profile. The function `PMSetProfile` is useful only if the graphics context is QuickDraw and the current port is the printing port.

You should call this function each time you want to change the profile used to draw page elements. The printing system resets the profile to the default at the beginning of each page. If you call the function `PMSetProfile` a second time, the old profile is ignored.

Special Considerations

This function is deprecated because QuickDraw is deprecated. When drawing with Quartz, the current stroke and fill color space and the color space associated with an image are used to characterize color. Quartz provides ways to use ColorSync profiles to create color spaces, so you can characterize color using ColorSync simply by drawing with Quartz.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSetResolution

Sets the application drawing resolution. (Deprecated in Mac OS X v10.5. Draw using Quartz 2D and call `CGContextScaleCTM` (page 105) instead.)

```
OSStatus PMSetResolution (
   PMPageFormat pageFormat,
   const PMResolution *res
);
```

Parameters

pageFormat
The page format object whose drawing resolution you want to set.

res
A pointer to a structure of type `PMResolution` (page 2278) that specifies the desired drawing resolution for your application. You should specify the best resolution for your data. The printing system handles the mapping between the resolution you specify and the printer resolution.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
If you call this function after initiating a print job, the change is ignored for the current job.

Special Considerations

This function was needed in the past because QuickDraw uses integer coordinates and has no notion of scaling coordinate systems. For Quartz drawing, this function is obsolete. To change the resolution, draw with fractional coordinates or scale the coordinate system and draw with integer coordinates.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMSetScale

Sets the scaling factor for the page and paper rectangles.

```
OSStatus PMSetScale (
   PMPageFormat pageFormat,
   double scale
);
```

Parameters

pageFormat

The page format object whose scaling factor you want to set.

scale

The desired scaling factor expressed as a percentage. For example, for 50 percent scaling, pass a value of 50.0; for no scaling, pass 100.0.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
You can call the function `PMSetScale` to change the scaling factor that appears when your application invokes the Page Setup dialog.

If you call `PMSetScale` after calling `PMSessionPageSetupDialog`, make sure you call `PMSessionValidatePageFormat` (page 2248) before you call `PMSessionBeginCGDocument` or `PMSessionBeginDocument`.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.0 and later.

See Also
`PMGetScale` (page 2170)

Declared In
`PMCore.h`

PMSetUnadjustedPaperRect

Requests a particular paper size, unaffected by rotation, resolution, or scaling. (Deprecated in Mac OS X v10.5. To set a particular paper size, obtain or create a `PMPaper` (page 2275) object and call `PMCreatePageFormatWithPMPaper` (page 2144).)

```
OSStatus PMSetUnadjustedPaperRect (
   PMPageFormat pageFormat,
   const PMRect *paperRect
);
```

Parameters

pageFormat

The page format object whose unadjusted paper rectangle you want to set.

paperRect

A pointer to a structure of type `PMRect` that specifies the desired paper size, in points. The coordinates of the upper-left corner of the paper rectangle are specified relative to the page rectangle. See *Supporting Printing in Your Carbon Application* for more information on page and paper rectangles.

Return Value
A result code. See "Core Printing Result Codes" (page 2298). The result code `kPMValueOutOfRange` indicates that the printer driver does not support the requested page size.

Discussion
This function is not recommended. You should call this function only if your application provides desktop publishing and the Page Setup dialog does not provide sufficient control. Typically, such applications display their own specialized document format dialog.

If you decide to use this function, you must call it between the creation and release of a printing session. After using the function `PMSetUnadjustedPaperRect` you should always call `PMSessionValidatePageFormat` (page 2248) then call `PMGetUnadjustedPaperRect` (page 2171) to verify that the paper size you set is recorded by the printer driver.

If you call this function after initiating a print job, the change is ignored for the current job.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMUnflattenPageFormat

Rebuilds a page format object from a Memory Manager handle that contains flattened page format data. (Deprecated in Mac OS X v10.5. Use `PMPageFormatCreateWithDataRepresentation` (page 2175) instead.)

```
OSStatus PMUnflattenPageFormat (
   Handle flatFormat,
   PMPageFormat *pageFormat
);
```

Parameters

flatFormat
A handle to a previously flattened page format object. You are responsible for disposing of the handle.

pageFormat
A pointer to your `PMPageFormat` (page 2275) variable. On return, the variable refers to a page format object that contains the data retrieved from the flattened page format data. You are responsible for releasing the page format object with the function `PMRelease` (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298). The result code `kPMInvalidParameter` is returned if the flattened `PMPageFormat` object was created by an incompatible version of Core Printing.

Discussion
The `PMUnflattenPageFormat` function creates a new `PMPageFormat` object that contains the data from the flattened page format data. You should call the function `PMSessionValidatePageFormat` (page 2248) to make sure the page format object contains valid values.

If the function returns the result code `kPMInvalidParameter` you need to create a new, default page format object. You should also notify the user that the flattened page format is not valid.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMUnflattenPageFormatWithCFData

Rebuilds a page format object from a Core Foundation data object that contains flattened page format data. (Deprecated in Mac OS X v10.5. Use `PMPageFormatCreateWithDataRepresentation` (page 2175) instead.)

```
OSStatus PMUnflattenPageFormatWithCFData (
   CFDataRef flattenCFData,
   PMPageFormat *pageFormat
);
```

Parameters

`flattenCFData`
A Core Foundation data object that contains a flattened representation of a page format object.

`pageFormat`
A pointer to your `PMPageFormat` (page 2275) variable. On return, the variable refers to a page format object that is rebuilt from the specified Core Foundation data object. You are responsible for releasing the page format object with the function `PMRelease` (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.5.

See Also
`PMFlattenPageFormatToCFData` (page 2149)

Declared In
`PMCoreDeprecated.h`

PMUnflattenPageFormatWithURL

Rebuilds a page format object from a file system URL that contains flattened page format data. (Deprecated in Mac OS X v10.5. Instead read the data into a CFData object and use `PMPageFormatCreateWithDataRepresentation` (page 2175).)

```
OSStatus PMUnflattenPageFormatWithURL (
   CFURLRef flattenFileURL,
   PMPageFormat *pageFormat
);
```

Parameters

flattenFileURL

A Core Foundation URL that specifies a file containing a flattened representation of a page format object.

pageFormat

A pointer to your PMPageFormat (page 2275) variable. On return, the variable refers to a page format object that is rebuilt from the specified file. You are responsible for releasing the page format object with the function PMRelease (page 2214).

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.4 and later.

Deprecated in Mac OS X v10.5.

See Also

PMFlattenPageFormatToURL (page 2150)

Declared In

PMCoreDeprecated.h

PMUnflattenPrintSettings

Rebuilds a print settings object from a Memory Manager handle that contains flattened print settings data. (Deprecated in Mac OS X v10.5. Use PMPrintSettingsCreateWithDataRepresentation (page 2209) instead.)

```
OSStatus PMUnflattenPrintSettings (
   Handle flatSettings,
   PMPrintSettings *printSettings
);
```

Parameters

flatSettings

A handle to a flattened representation of a print settings object.

printSettings

A pointer to your PMPrintSettings (page 2277) variable. On return, the variable refers to a print settings object that contains the data retrieved from the flattened print settings. You are responsible for releasing the print settings object with the function PMRelease (page 2214).

Return Value

A result code. See "Core Printing Result Codes" (page 2298). The result code kPMInvalidParameter is returned if the flattened PMPrintSettings object was created by an incompatible version of Core Printing.

Discussion

The PMUnflattenPrintSettings function creates a new PMPrintSettings object containing the data from the flattened print settings. You should call the function PMSessionValidatePrintSettings (page 2249), as some values in the print settings object may no longer be valid.

If the function returns the result code `kPMInvalidParameter` you need to create a new, default print settings object. You should also notify the user that the print settings are not valid.

There are no scoping requirements as to when you may use this function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMUnflattenPrintSettingsWithCFData

Rebuilds a print settings object from a Core Foundation data object that contains flattened print settings data. (Deprecated in Mac OS X v10.5. Use `PMPrintSettingsCreateWithDataRepresentation` (page 2209) instead.)

```
OSStatus PMUnflattenPrintSettingsWithCFData (
   CFDataRef flattenCFData,
   PMPrintSettings *printSettings
);
```

Parameters

`flattenCFData`
A flattened representation of a print settings object.

`printSettings`
A pointer to your `PMPrintSettings` (page 2277) variable. On return, the variable refers to a print settings object rebuilt from the specified Core Foundation data object. You are responsible for releasing the print settings object with the function `PMRelease` (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.5.

See Also
`PMFlattenPrintSettingsToCFData` (page 2151)

Declared In
`PMCoreDeprecated.h`

PMUnflattenPrintSettingsWithURL

Rebuilds a print settings object from a file that contains flattened print settings data. (Deprecated in Mac OS X v10.5. Instead read the data into a CFData object and use `PMPrintSettingsCreateWithDataRepresentation` (page 2209).)

```
OSStatus PMUnflattenPrintSettingsWithURL (
   CFURLRef flattenFileURL,
   PMPrintSettings *printSettings
);
```

Parameters

flattenFileURL

A file containing a flattened representation of a print settings object.

printSettings

A pointer to your PMPrintSettings (page 2277) variable. On return, the variable refers to a print settings object rebuilt from the specified file. You are responsible for releasing the print settings object with the function PMRelease (page 2214).

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.5.

See Also
PMFlattenPrintSettingsToURL (page 2152)

Declared In
PMCoreDeprecated.h

PMValidatePageFormat

Obtains a valid PMPageFormat object. (Deprecated in Mac OS X v10.4. Use PMSessionValidatePageFormat (page 2248) instead.)

```
OSStatus PMValidatePageFormat (
    PMPageFormat pageFormat,
    Boolean *result
);
```

Parameters

pageFormat

A PMPageFormat object to be validated.

result

Returns true if any parameters were changed, false if no changes were required.

Return Value
A result code. See "Core Printing Result Codes" (page 2298).

Discussion
Valid after calling PMBegin and creating a page format object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMCoreDeprecated.h

PMValidatePrintSettings

Obtains a valid `PMPrintSettings` object. (Deprecated in Mac OS X v10.4. Use `PMSessionValidatePrintSettings` (page 2249) instead.)

```
OSStatus PMValidatePrintSettings (
    PMPrintSettings printSettings,
    Boolean *result
);
```

Parameters

printSettings

The `PMPrintSettings` object to be validated.

result

On return, a value of `true` if any parameters were changed, or `false` if no changes were required.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

Valid after calling `PMBegin` and creating a print settings object.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PMCoreDeprecated.h`

PMWorkflowCopyItems

Obtains an array of the available PDF workflow items.

```
OSStatus PMWorkflowCopyItems (
   CFArrayRef *workflowItems
);
```

Parameters

workflowItems

A pointer to your `CFArrayRef` variable. On return, the variable refers to an Core Foundation array. Each element in the array is a dictionary that describes either a PDF workflow item or a folder containing a set of PDF workflow items. For a list of possible keys, see "`PDF Workflow Dictionary Keys`" (page 2284). You are responsible for releasing the array.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`PMCore.h`

PMWorkflowSubmitPDFWithOptions

Submits a PDF file for workflow processing using the specified CUPS options string.

```
OSStatus PMWorkflowSubmitPDFWithOptions (
   CFURLRef workflowItem,
   CFStringRef title,
   const char *options,
   CFURLRef pdfFile
);
```

Parameters

workflowItem

A file system URL pointing to the workflow item that will handle the PDF file. See `PMWorkflowCopyItems` (page 2270). The following table describes the different types of workflow items for this function.

Workflow item	Description
Automator action	The action is executed for the PDF file. Available in Mac OS X v10.4 and later.
Folder alias	The PDF file is moved to the resolved folder.
Application or application alias	The application is sent an open event along with a reference to the PDF file.
Compiled AppleScript	The script is run with an open event along with a reference to the PDF file.
Executable tool	The tool is run with the following parameters: *title*, *options*, and *pdfFile*.

title

The user-displayable name of the PDF document.

options

A string of CUPS-style key-value pairs that may be passed to the PDF workflow item. This parameter can be `NULL` in which case an empty string of options is used.

pdfFile

A file system URL pointing to the PDF file to be processed by the workflow item.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

The printing system uses this function in conjunction with the function `PMWorkflowCopyItems` (page 2270) to implement the PDF workflow button in the Print dialog.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`PMCore.h`

PMWorkflowSubmitPDFWithSettings

Submits a PDF file for workflow processing using the specified print settings.

```
OSStatus PMWorkflowSubmitPDFWithSettings (
   CFURLRef workflowItem,
   PMPrintSettings settings,
   CFURLRef pdfFile
);
```

Parameters

workflowItem

A file system URL pointing to the workflow item that will handle the PDF file. See `PMWorkflowCopyItems` (page 2270). The following table describes the different types of workflow items for this function.

Workflow item	Description
Automator action	The action is executed for the PDF file. Available in Mac OS X v10.4 and later.
Folder alias	The PDF file is moved to the resolved folder.
Application or application alias	The application is sent an open event along with a reference to the PDF file.
Compiled AppleScript	The script is run with an open event along with a reference to the PDF file.
Executable tool	The tool is run with the specified settings and PDF file. This function converts these parameters into a CUPS options string and passes the options string to the tool.

settings

The print settings to apply to the PDF document. These settings are passed to the workflow item as a CUPS options string.

pdfFile

A file system URL pointing to the PDF file to be processed by the workflow item.

Return Value

A result code. See "Core Printing Result Codes" (page 2298).

Discussion

The printing system uses this function in conjunction with the function `PMWorkflowCopyItems` (page 2270) to implement the PDF workflow button in the Print dialog.

Special Considerations

In Mac OS X v10.4 and earlier, this function is not implemented and returns an error. You can use the function `PMWorkflowSubmitPDFWithOptions` (page 2271) together with the function `PMPrintSettingsToOptions` (page 2213) instead.

Availability

Available in Mac OS X v10.3 and later.

Declared In
`PMCore.h`

Callbacks

PMIdleProcPtr

Defines a pointer to an idle function. (**Deprecated.** There is no replacement; this callback function was included to facilitate porting legacy applications to Mac OS X, but it serves no useful purpose.)

```
typedef void (*PMIdleProcPtr) (void);
```

You would declare your idle function like this if you were to name it `MyPrintIdleCallback`:

```
void MyPrintIdleCallback (void);
```

Discussion
If you install an idle function using the function `PMSessionSetIdleProc`, the printing system calls your idle function periodically during your print loop. Your idle function can display application status while printing, but it should not duplicate information displayed by the printing system or the printer driver. If you don't install an idle function, you get the standard dialog for the current driver in Mac OS 8 and 9.

Your idle function must check whether the user has pressed Command-period, in which case your application should stop its printing operation. If your status dialog contains a button to cancel the printing operation, your idle function should also check for clicks in the button and respond accordingly.

To provide a pointer to your idle function, you create a universal procedure pointer (UPP) of type `PMIdleUPP`, using the function `NewPMIdleUPP`. You can do so with code similar to the following:

```
PMIdleUPP MyPrintIdleUPP;
MyPrintIdleUPP = NewPMIdleUPP (&MyPrintIdleCallback);
```

When your print job is completed, you should use the function `DisposePMIdleUPP` to dispose of the universal procedure pointer associated with your idle function. However, if you will use the same idle function in subsequent print jobs, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Special Considerations

Your idle function is not called in Mac OS X. It's only called in Mac OS 8 and 9.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

Data Types

PMDialog

An opaque type that represents a custom printing dialog.

```
typedef struct OpaquePMDialog* PMDialog;
```

Discussion
This data type is used by functions that are not recommended or deprecated.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMDefinitionsDeprecated.h`

PMIdleUPP

A type that defines a universal procedure pointer to an idle callback.

```
typedef PMIdleProcPtr PMIdleUPP;
```

Discussion
This data type is used by functions that are not recommended or deprecated.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMCoreDeprecated.h`

PMLanguageInfo

A data structure that contains level, version, and release information for the imaging language used by a printer driver.

```
struct PMLanguageInfo {
    Str32 level;
    Str32 version;
    Str32 release;
};
```

Fields
`level`
Specifies the level of the imaging language used by the printer driver.
`version`
Specifies the version of the imaging language.

`release`
: Specifies the release of the imaging language.

PMObject

The base type for all the opaque types used in Core Printing.

```
typedef const void* PMObject;
```

Discussion
`PMObject` is the base type for opaque types such as `PMPrintSession`, `PMPageFormat`, `PMPrintSettings`, `PMPrinter`, `PMPaper`, `PMPreset`, and `PMServer`. `PMObject` is used in functions such as `PMRetain` (page 2215) and `PMRelease` (page 2214) that operate on any opaque type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMDefinitions.h`

PMPageFormat

An opaque type that stores the settings in the Page Setup dialog.

```
typedef struct OpaquePMPageFormat* PMPageFormat;
```

Discussion
Your application uses page format objects to store information such as the paper size, orientation, and scale of pages in a printing session. To create a page format object, you use the function `PMCreatePageFormat` (page 2143). A new page format object is empty and unusable until you call `PMSessionDefaultPageFormat` (page 2225) or `PMCopyPageFormat` (page 2141) to initialize the settings. You can also use the functions `PMSetPageFormatExtendedData` (page 2258) and `PMGetPageFormatExtendedData` (page 2164) to store and retrieve application-specific data in a page format object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMDefinitions.h`

PMPaper

An opaque type that stores information about the paper used in a print job.

```
typedef struct OpaquePMPaper* PMPaper;
```

Discussion
Your application uses paper objects to identify standard and custom types of printing paper.

Availability
Available in Mac OS X v10.3 and later.

Declared In
PMDefinitions.h

PMPaperMargins

A data structure that specifies the unprintable area of a paper object.

```
typedef PMRect PMPaperMargins;
```

Discussion
Your application specifies paper margins when calling the function PMPaperCreateCustom (page 2177) to create a custom paper type. You can obtain a paper's margins with the function PMPaperGetMargins (page 2180).

Availability
Available in Mac OS X v10.3 and later.

Declared In
PMDefinitions.h

PMPreset

An opaque type that stores information about a named preset available for a print job.

```
typedef struct OpaquePMPreset* PMPreset;
```

Discussion
Your application uses a preset object to identify a named preset in the Print dialog. You typically obtain an instance of this type using the function PMPrinterCopyPresets (page 2189).

Availability
Available in Mac OS X v10.3 and later.

Declared In
PMDefinitions.h

PMPrinter

An opaque type that represents a printer.

```
typedef struct OpaquePMPrinter* PMPrinter;
```

Discussion
You typically obtain a printer object using the function PMSessionGetCurrentPrinter (page 2231) or PMServerCreatePrinterList (page 2215).

Availability
Available in Mac OS X v10.0 and later.

Declared In
PMDefinitions.h

PMPrintSession

An opaque type that stores information about a print job.

```
typedef struct OpaquePMPrintSession* PMPrintSession;
```

Discussion
A printing session object contains information that's needed by the page format and print settings objects, such as default page format and print settings values. For this reason, some printing functions can be called only after you have created a printing session object. For example, setting defaults for or validating page format and print settings objects can only be done after you have created a printing session object. Your application creates a printing session object using the function `PMCreateSession` (page 2145).

You can use a printing session to implement multithreaded printing, and you can create multiple sessions within a single-threaded application. If your application does not use sheets, then your application can open only one dialog at a time. Each printing session can have its own dialog, and settings changed in one dialog are independent of settings in any other dialog.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMDefinitions.h`

PMPrintSettings

An opaque type that stores the settings in the Print dialog.

```
typedef struct OpaquePMPrintSettings* PMPrintSettings;
```

Discussion
Your application uses print settings objects to store information such as the number of copies and the range of pages to print in a printing session. To create a print settings object, you use the function `PMCreatePrintSettings` (page 2144). A new print settings object is empty and unusable until you call `PMSessionDefaultPrintSettings` (page 2225) or `PMCopyPrintSettings` (page 2142) to initialize the settings. You can also use the functions `PMSetPrintSettingsExtendedData` (page 2261) and `PMGetPrintSettingsExtendedData` (page 2168) to store and retrieve application-specific data in a print settings object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMDefinitions.h`

PMRect

A data structure that describes a rectangle using four double-precision coordinates.

```
struct PMRect {
    double top;
    double left;
    double bottom;
    double right;
};
```

Fields

`top`
The vertical coordinate for the upper-left point of the rectangle.

`left`
The horizontal coordinate for the upper-left point of the rectangle.

`bottom`
The vertical coordinate for the lower-right point of the rectangle.

`right`
The horizontal coordinate for the lower-right point of the rectangle.

PMResolution

A data structure that contains printing resolution information.

```
struct PMResolution {
    double hRes;
    double vRes;
};
```

Fields

`hRes`
The horizontal resolution in dots per inch (dpi).

`vRes`
The vertical resolution in dots per inch (dpi).

Discussion
The functions `PMGetResolution` (page 2169) and `PMPrinterGetPrinterResolution` (page 2199) use this structure to return printing resolution information. Your application can pass this information to the function `PMSetResolution` (page 2263).

PMServer

An opaque type that identifies a local or remote print server.

```
typedef struct OpaquePMServer* PMServer;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`PMDefinitions.h`

PMPrintContext

An opaque type that describes the graphics environment for printing a document.

```
typedef struct OpaquePMPrintContext* PMPrintContext;
```

Discussion
This data type is used by functions that are not recommended or deprecated.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMDefinitionsDeprecated.h`

PMColorMode

A type that specifies color modes to use for printing.

```
typedef UInt16 PMColorMode;
```

Discussion
This data type is used by functions that are not recommended or deprecated.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMDefinitionsDeprecated.h`

Constants

Data Not Wanted Constants

Constants your application can use to indicate it does not need certain types of data returned by various printing functions.

```
#define kPMNoData  NULL
#define kPMDontWantSize  NULL
#define kPMDontWantData  NULL
#define kPMDontWantBoolean  NULL
#define kPMNoPrintSettings  NULL
#define kPMNoPageFormat  NULL
#define kPMNoReference  NULL
```

Constants
`kPMNoData`
Specifies that your application does not need data returned for a particular parameter. For future compatibility, you are encouraged to use one of the following constants in cases where a specific type of data is not required.

`kPMDontWantSize`
Specifies that your application does not need the size information returned by the printing function.

`kPMDontWantData`
Specifies that your application does not need the data returned by the printing function.

`kPMDontWantBoolean`
Specifies that your application does not need a Boolean value returned by the printing function.

`kPMNoPrintSettings`
Specifies that your application does not need a `PMPrintSettings` object returned by the printing function.

`kPMNoPageFormat`
Specifies that your application does not need a `PMPageFormat` object returned by the printing function.

`kPMNoReference`
Specifies that your application does not need a reference returned by the printing function.

Data Representation Formats

Constants that specify the format of the data representation created with the functions `PMPageFormatCreateDataRepresentation` (page 2174) and `PMPrintSettingsCreateDataRepresentation` (page 2208).

```
enum PMDataFormat {
  kPMDataFormatXMLDefault = 0,
  kPMDataFormatXMLMinimal = 1,
  kPMDataFormatXMLCompressed = 2
};
typedef enum PMDataFormat PMDataFormat;
```

Constants

`kPMDataFormatXMLDefault`
Specifies a data format that is compatible with all Mac OS X versions. Data in this format can be used with the `PMUnflattenXXX` functions present in versions of Mac OS X prior to 10.5. This format is a pure XML representation of the data. However, this format is much larger than the more modern data formats described below.

Available in Mac OS X v10.5 and later.

Declared in `PMDefinitions.h`.

`kPMDataFormatXMLMinimal`
Specifies an uncompressed data format that is approximately 3-5 times smaller than `kPMDataFormatXMLDefault`. This data format is only compatible with Mac OS X v10.5 and later. This format is a good choice when you do not need to use the data in versions of Mac OS X prior to 10.5 and you need a pure XML representation of the data.

Available in Mac OS X v10.5 and later.

Declared in `PMDefinitions.h`.

`kPMDataFormatXMLCompressed`
: Specifies a compressed data format that is approximately 20 times smaller than `kPMDataFormatXMLDefault`. This data format is only compatible with Mac OS X v10.5 and later. This format is a good choice when you do not need to use the data in versions of Mac OS X prior to 10.5 and the minimum data size is important. Note that this format is not a pure XML representation of the data.

Available in Mac OS X v10.5 and later.

Declared in `PMDefinitions.h`.

Destination Types

Constants that specify a destination for a print job.

```
typedef UInt16 PMDestinationType;
enum {
    kPMDestinationInvalid = 0,
    kPMDestinationPrinter = 1,
    kPMDestinationFile = 2,
    kPMDestinationFax = 3,
    kPMDestinationPreview = 4,
    kPMDestinationProcessPDF = 5
};
```

Constants

`kPMDestinationInvalid`
: Specifies the destination is invalid.

Available in Mac OS X v10.1 and later.

Declared in `PMDefinitions.h`.

`kPMDestinationPrinter`
: Specifies output to a printer.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMDestinationFile`
: Specifies output to a file.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMDestinationFax`
: Specifies output to a fax. This destination is currently not supported.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMDestinationPreview`
: Specifies output to print preview.

Available in Mac OS X v10.1 and later.

Declared in `PMDefinitions.h`.

`kPMDestinationProcessPDF`
Specifies output to a PDF workflow option.

Available in Mac OS X v10.4 and later.

Declared in `PMDefinitions.h`.

Document Format Strings

Constants that specify the document format for a print job.

```
#define kPMDocumentFormatDefault
            CFSTR("com.apple.documentformat.default")
#define kPMDocumentFormatPDF
            CFSTR("application/pdf")
#define kPMDocumentFormatPICT
            CFSTR("application/vnd.apple.printing-pict")
#define kPMDocumentFormatPICTPS
            CFSTR("application/vnd.apple.printing-pict-ps")
#define kPMDocumentFormatPostScript
            CFSTR("application/postscript")
```

Constants

`kPMDocumentFormatDefault`
Specifies the default format for the printing system. In Mac OS X, the default format is PDF.

`kPMDocumentFormatPDF`
Specifies PDF.

`kPMDocumentFormatPICT`
Specifies PICT format.

`kPMDocumentFormatPICTPS`
Specifies PICT format with embedded PostScript.

`kPMDocumentFormatPostScript`
Specifies PostScript format.

Duplex Modes

Constants that specify duplex mode settings.

```
typedef UInt32 PMDuplexMode;
enum {
    kPMDuplexNone = 0x0001,
    kPMDuplexNoTumble = 0x0002,
    kPMDuplexTumble = 0x0003,
    kPMSimplexTumble = 0x0004
};
```

Constants

`kPMDuplexNone`
Print on only one side of the paper.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMDuplexNoTumble`
Print on both sides of the paper, with both sides oriented in the same direction (no tumbling.)

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMDuplexTumble`
Print on both sides of the paper, with the output on the second side flipped relative to the first side (tumbling on.)

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMSimplexTumble`
Print on only one side of the paper, but tumble the images while printing. This mode is not supported at this time.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

Graphics Context Types

Constants that specify the graphics context for a print job.

```
#define kPMGraphicsContextDefault
        CFSTR("com.apple.graphicscontext.default")
#define kPMGraphicsContextQuickdraw
        CFSTR("com.apple.graphicscontext.quickdraw")
#define kPMGraphicsContextCoreGraphics
        CFSTR("com.apple.graphicscontext.coregraphics")
```

Constants

`kPMGraphicsContextDefault`
Specifies the default graphics context for the application's runtime environment.

`kPMGraphicsContextQuickdraw`
Specifies a QuickDraw graphics context.

`kPMGraphicsContextCoreGraphics`
Specifies a Quartz graphics context. The default coordinate system for a Quartz printing context is not the same as that used for a QuickDraw printing context. A Quartz printing context defines the coordinates of the lower-left corner of the paper as (0,0) whereas the origin for a QuickDraw is at the upper-right corner of the paper's imageable area.

Page Orientation Constants

Constants that specify page orientation.

```
typedef UInt16 PMOrientation;
enum {
    kPMPortrait = 1,
    kPMLandscape = 2,
    kPMReversePortrait = 3,
    kPMReverseLandscape = 4
};
```

Constants

`kPMPortrait`

Specifies portrait (vertical) page orientation. Portrait orientation performs no alteration of the logical page.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMLandscape`

Specifies landscape (horizontal) orientation. Landscape orientation performs a 90° counterclockwise rotation on the logical page image and sets the Quartz origin to the upper-left corner of the unrotated logical page with positive y-values increasing across and to the right of the unrotated page. This has the effect of rotating the logical page image 90° clockwise. In other words, the image appears on the unrotated page as if it were rotated 90° clockwise.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMReversePortrait`

Specifies reverse portrait orientation. Reverse portrait orientation performs a 180° rotation on the logical page rectangle and sets the Quartz origin to the upper-right corner of the unrotated logical page with positive y-values increasing downwards. This has the effect of rotating the logical page image 180°. Reverse portrait orientation is supported in Mac OS X v10.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

`kPMReverseLandscape`

Specifies reverse landscape page orientation. Landscape orientation performs a 90° clockwise rotation on the logical page rectangle and sets the Quartz origin to the lower-right corner of the unrotated logical page, with the positive y-values increasing to across and to the left of the unrotated page. This has the effect of rotating the logical page image 90° counterclockwise. In other words, the image appears on the unrotated page as if it were rotated 90° counterclockwise.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

PDF Workflow Dictionary Keys

Constants that specify the keys in a PDF workflow dictionary.

```
#define kPDFWorkFlowItemURLKey CFSTR("itemURL")
#define kPDFWorkflowDisplayNameKey CFSTR("displayName")
#define kPDFWorkflowFolderURLKey CFSTR("folderURL")
#define kPDFWorkflowItemsKey CFSTR("items")
```

Constants

`kPDFWorkFlowItemURLKey`
The URL to the PDF workflow item.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

`kPDFWorkflowDisplayNameKey`
The user-displayable name for the PDF workflow item.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

`kPDFWorkflowFolderURLKey`
The URL to the folder containing PDF workflow items.

Available in Mac OS X v10.5 and later.

Declared in `PMDefinitions.h`.

`kPDFWorkflowItemsKey`
A Core Foundation array describing the PDF workflow items in the folder.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

PostScript Injection Dictionary Keys

Constants that specify keys for PostScript injection dictionary entries.

```
#define kPSInjectionSectionKey     CFSTR("section")
#define kPSInjectionSubSectionKey  CFSTR("subsection")
#define kPSInjectionPageKey        CFSTR("page")
#define kPSInjectionPlacementKey   CFSTR("place")
#define kPSInjectionPostScriptKey  CFSTR("psdata")
```

Constants

`kPSInjectionSectionKey`
Specifies a section.

`kPSInjectionSubSectionKey`
Specifies a subsection.

`kPSInjectionPageKey`
Specifies a page.

`kPSInjectionPlacementKey`
Specifies placement.

`kPSInjectionPostScriptKey`
Specifies PostScript data.

PostScript Page Injection Options

Constants that specify PostScript injection options.

```
enum {
    kPSPageInjectAllPages = -1,
    kPSInjectionMaxDictSize = 5
};
```

Constants

`kPSPageInjectAllPages`

Specifies to inject all pages in the print job with PostScript code.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kPSInjectionMaxDictSize`

Specifies the maximum size needed for a dictionary used for PostScript injection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

PostScript Injection Placement Options

Constants that specify where in the print job to inject PostScript code.

```
typedef UInt16 PSInjectionPlacement;
enum {
    kPSInjectionBeforeSubsection = 1,
    kPSInjectionAfterSubsection = 2,
    kPSInjectionReplaceSubsection = 3
};
```

Constants

`kPSInjectionBeforeSubsection`

Specifies that your PostScript code be inserted before the standard PostScript code that is normally emitted for the subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kPSInjectionAfterSubsection`

Specifies that your PostScript code be inserted after the standard PostScript code that is normally emitted for the subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kPSInjectionReplaceSubsection`

Specifies that your PostScript code replace the standard PostScript code that is normally emitted for the subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

PostScript Injection Sections

Constants that specify keys for PostScript injection section values.

```
typedef SInt32 PSInjectionSection;
enum {
    kInjectionSectJob = 1,
    kInjectionSectCoverPage = 2
};
```

Constants

`kInjectionSectJob`

Specifies the job section. This is the default section if you do not specify a section key explicitly.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSectCoverPage`

Specifies the cover page section. Currently unsupported.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

PostScript Injection Subsections

Constants that specify PostScript injection values for the subsection key.

```
typedef SInt32 PSInjectionSubsection;
enum {
    kInjectionSubPSAdobe = 1,
    kInjectionSubPSAdobeEPS = 2,
    kInjectionSubBoundingBox = 3,
    kInjectionSubEndComments = 4,
    kInjectionSubOrientation = 5,
    kInjectionSubPages = 6,
    kInjectionSubPageOrder = 7,
    kInjectionSubBeginProlog = 8,
    kInjectionSubEndProlog = 9,
    kInjectionSubBeginSetup = 10,
    kInjectionSubEndSetup = 11,
    kInjectionSubBeginDefaults = 12,
    kInjectionSubEndDefaults = 13,
    kInjectionSubDocFonts = 14,
    kInjectionSubDocNeededFonts = 15,
    kInjectionSubDocSuppliedFonts = 16,
    kInjectionSubDocNeededRes = 17,
    kInjectionSubDocSuppliedRes = 18,
    kInjectionSubDocCustomColors = 19,
    kInjectionSubDocProcessColors = 20,
    kInjectionSubPlateColor = 21,
    kInjectionSubPageTrailer = 22,
    kInjectionSubTrailer = 23,
    kInjectionSubEOF = 24,
    kInjectionSubBeginFont = 25,
    kInjectionSubEndFont = 26,
    kInjectionSubBeginResource = 27,
    kInjectionSubEndResource = 28,
    kInjectionSubPage = 29,
    kInjectionSubBeginPageSetup = 30,
    kInjectionSubEndPageSetup = 31
};
```

Constants

`kInjectionSubPSAdobe`

Specifies the "%!PS-Adobe" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubPSAdobeEPS`

Specifies the "%!PS-Adobe-3.0 EPSF-3.0" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubBoundingBox`

Specifies the "%BoundingBox" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEndComments`
Specifies the "%EndComments" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubOrientation`
Specifies the "%Orientation" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubPages`
Specifies the "%Pages" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubPageOrder`
Specifies the "%PageOrder" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubBeginProlog`
Specifies the "%BeginProlog" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEndProlog`
Specifies the "%EndProlog" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubBeginSetup`
Specifies the "%BeginSetup" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEndSetup`
Specifies the "%EndSetup" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubBeginDefaults`
Specifies the "%BeginDefaults" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEndDefaults`
Specifies the "%EndDefaults" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubDocFonts`
Specifies the "%DocumentFonts" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubDocNeededFonts`
Specifies the "%DocumentNeededFonts" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubDocSuppliedFonts`
Specifies the "%DocumentSuppliedFonts" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubDocNeededRes`
Specifies the "%DocumentNeededResources" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubDocSuppliedRes`
Specifies the "%DocumentSuppliedResources" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubDocCustomColors`
Specifies the "%DocumentCustomColors" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubDocProcessColors`
Specifies the "%DocumentProcessColors" subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubPlateColor`
Specifies the "%PlateColor" subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubPageTrailer`
Specifies the "%PageTrailer" subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubTrailer`
Specifies the "%Trailer" subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEOF`
Specifies the "%EOF" (end of file) subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubBeginFont`
Specifies the "%BeginFont" subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEndFont`
Specifies the "%EndFont" subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubBeginResource`
Specifies the "%BeginResource" subsection.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEndResource`
Specifies the "%EndResource" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubPage`
Specifies the "%Page" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubBeginPageSetup`
Specifies the "%BeginPageSetup" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kInjectionSubEndPageSetup`
Specifies the "%EndPageSetup" subsection.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

PostScript Printer Description File Domains

Constants that specify the domains for PostScript printer description (PPD) files.

```
typedef UInt16 PMPPDDomain;
enum {
    kAllPPDDomains = 1,
    kSystemPPDDomain = 2,
    kLocalPPDDomain = 3,
    kNetworkPPDDomain = 4,
    kUserPPDDomain = 5,
    kCUPSPPDDomain = 6
};
```

Constants

`kAllPPDDomains`
Specifies all available domains.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

`kSystemPPDDomain`
Specifies the system domain.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

`kLocalPPDDomain`
Specifies the local domain.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

`kNetworkPPDDomain`
Specifies the network domain.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

`kUserPPDDomain`
Specifies the user domain.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

`kCUPSPPDDomain`
Specifies the CUPS domain.

Available in Mac OS X v10.3 and later.

Declared in `PMDefinitions.h`.

Print All Pages Constant

A constant that specifies that all pages of a document should be printed.

```
enum {
    kPMPrintAllPages = -1
};
```

Constants

`kPMPrintAllPages`
Specifies that all pages of a document should be printed.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

Print Quality Modes

Constants that specify standard options for print quality.

```
typedef UInt32 PMQualityMode;
enum {
    kPMQualityLowest = 0,
    kPMQualityInkSaver = 1,
    kPMQualityDraft = 4,
    kPMQualityNormal = 8,
    kPMQualityPhoto = 11,
    kPMQualityBest = 13,
    kPMQualityHighest = 15
};
```

Constants

`kPMQualityLowest`
Specifies to use the lowest print quality available to the printer.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMQualityInkSaver`
Specifies to use a mode that saves ink, even if it slows printing.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMQualityDraft`
Specifies to print at the highest speed, with the amount of ink used as a secondary consideration.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMQualityNormal`
Specifies a general usage mode that balances quality and speed.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMQualityPhoto`
Specifies to optimize the quality of photos on the page, with speed not a concern.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMQualityBest`
Specifies to get the best print quality for all objects and photos on a page.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

`kPMQualityHighest`
Specifies to use the highest print quality available to the printer.
Available in Mac OS X v10.0 and later.
Declared in `PMDefinitions.h`.

Print Queue States

Constants that specify the current state of a print queue.

```
typedef UInt16 PMPrinterState;
enum {
    kPMPrinterIdle = 3,
    kPMPrinterProcessing = 4,
    kPMPrinterStopped = 5
};
```

Constants

`kPMPrinterIdle`

Specifies the idle state.

Available in Mac OS X v10.2 and later.

Declared in `PMDefinitions.h`.

`kPMPrinterProcessing`

Specifies the processing state.

Available in Mac OS X v10.2 and later.

Declared in `PMDefinitions.h`.

`kPMPrinterStopped`

Specifies the stopped state.

Available in Mac OS X v10.2 and later.

Declared in `PMDefinitions.h`.

Printer Description Types

Constants that specify printer description types.

```
#define kPMPPDDescriptionType CFSTR("PMPPDDescriptionType")
```

Constants

`kPMPPDDescriptionType`

Specifies a PostScript printer description (PPD).

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

Tag Constants

Constants that specify values, such as minimum and maximum values, that your application can pass to or obtain from printing functions.

```
typedef UInt32 PMTag;
enum {
    kPMCurrentValue = 'curr',
    kPMDefaultValue = 'dflt',
    kPMMinimumValue = 'minv',
    kPMMaximumValue = 'maxv',
    kPMSourceProfile = 'srcp',
    kPMMinRange = 'mnrg',
    kPMMaxRange = 'mxrg',
    kPMMinSquareResolution = 'mins',
    kPMMaxSquareResolution = 'maxs',
    kPMDefaultResolution = 'dftr'
};
```

Constants

`kPMCurrentValue`
Specifies the current setting or value.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMDefaultValue`
Specifies the default setting or value.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMMinimumValue`
Specifies the minimum setting or value.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMMaximumValue`
Specifies the maximum setting or value.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMSourceProfile`
Specifies a ColorSync source profile.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMMinRange`
Specifies the minimum resolution supported by the printer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMMaxRange`
Specifies the maximum resolution supported by the printer.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kPMMinSquareResolution`
Specifies the minimum resolution setting for which the horizontal and vertical resolutions are equal.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kPMMaxSquareResolution`
Specifies the maximum resolution setting for which the horizontal and vertical resolutions are equal.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

`kPMDefaultResolution`
Specifies the default resolution setting for the printer (typically 72 dots per inch).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMDefinitionsDeprecated.h`.

User Cancellation Constant

A constant that specifies an error value that indicates the user canceled a printing operation.

```
enum {
    kPMCancel = 128
};
```

Constants

`kPMCancel`
Specifies that the user clicked the Cancel button in a Print or Page Setup dialog.

Available in Mac OS X v10.0 and later.

Declared in `PMDefinitions.h`.

Discussion

This constant is provided for compatibility with old applications and printer drivers that expect the `iPrAbort` error code to be returned when the user cancels a printing operation.

The default idle function checks for Command-period keyboard events during printing, and sets the error condition equal to `kPMCancel` if one occurs. Your application can check for this condition using the `PMSessionError` function, and should cancel the print job if `kPMCancel` is returned.

If you supply your own idle function (not needed in Mac OS 8, 9, or X), your function must check for Command-period keyboard events, and set the error condition using the function `PMSessionSetError`.

Color Modes

Constants that specify a color mode to use for printing.

```
typedef UInt16 PMColorMode;
enum {
    kPMBlackAndWhite = 1,
    kPMGray = 2,
    kPMColor = 3,
    kPMColorModeDuotone = 4,
    kPMColorModeSpecialColor = 5
};
```

Constants

`kPMBlackAndWhite`
Specifies black-and-white mode.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMGray`
Specifies grayscale mode.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMColor`
Specifies color mode.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMColorModeDuotone`
Specifies two-channel color mode.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

`kPMColorModeSpecialColor`
Specifies to allow special colors such as metallic and light cyan.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `PMDefinitionsDeprecated.h`.

Discussion
These constants are used by functions that are deprecated.

Result Codes

This table lists the result codes defined for Core Printing.

Result Code	Value	Description
kPMGeneralError	-30870	An unspecified error occurred. Available in Mac OS X v10.0 and later.
kPMOutOfScope	-30871	Your application called this function out of sequence with other printing functions. Available in Mac OS X v10.0 and later.
kPMNoDefaultPrinter	-30872	The user has not specified a default printer. Available in Mac OS X v10.0 and later.
kPMNotImplemented	-30873	The function is not implemented. Available in Mac OS X v10.0 and later.
kPMNoSuchEntry	-30874	There is no entry to match your application's request. Available in Mac OS X v10.0 and later.
kPMInvalidPrintSettings	-30875	Your application passed an invalid print settings object. Available in Mac OS X v10.0 and later.
kPMInvalidPageFormat	-30876	Your application passed an invalid page format object. Available in Mac OS X v10.0 and later.
kPMValueOutOfRange	-30877	Your application passed an out-of-range value. Available in Mac OS X v10.0 and later.
kPMLockIgnored	-30878	The lock value was ignored. Available in Mac OS X v10.0 and later.
kPMInvalidPrintSession	-30879	Your application passed an invalid printing session object. Available in Mac OS X v10.0 and later.
kPMInvalidPrinter	-30880	Your application passed an invalid printer object. Available in Mac OS X v10.0 and later.
kPMObjectInUse	-30881	The specified object is in use. Available in Mac OS X v10.0 and later.
kPMInvalidIndex	-30882	An array index is invalid. Available in Mac OS X v10.0 and later.
kPMStringConversionFailure	-30883	An internal error occurred while converting a string. Available in Mac OS X v10.0 and later.
kPMXMLParseError	-30884	An error occurred while parsing XML data. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kPMInvalidJobTemplate	-30885	An internal error occurred while creating a job template. Available in Mac OS X v10.0 and later.
kPMInvalidPrinterInfo	-30886	The printer information is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidConnection	-30887	The printer connection type is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidKey	-30888	The key in a ticket, job template, or dictionary is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidValue	-30889	The value in a ticket, job template, or dictionary is missing. Available in Mac OS X v10.0 and later.
kPMInvalidAllocator	-30890	The specified memory allocator is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidTicket	-30891	The job ticket is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidItem	-30892	The item being added to a ticket is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidType	-30893	The data type in a ticket, job template, or dictionary is not the expected type. Available in Mac OS X v10.0 and later.
kPMInvalidReply	-30894	A remote server or client sent an invalid reply. Available in Mac OS X v10.0 and later.
kPMInvalidFileType	-30895	The file type is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidObject	-30896	The object is invalid. Available in Mac OS X v10.0 and later.
kPMInvalidPaper	-30897	Your application passed an invalid paper object. Available in Mac OS X v10.2 and later.
kPMInvalidCalibrationTarget	-30898	The dictionary specifying a printer calibration target is invalid. Available in Mac OS X v10.3 and later.
kPMInvalidPreset	-30899	Your application passed an invalid preset object. Available in Mac OS X v10.3 and later.

CGImageProperties Reference

Framework: ApplicationServices/ImageIO

Declared in CGImageProperties.h

Overview

CGImageProperties Reference defines constants that represent characteristics of images used by the Image I/O framework.

Constants

Format-Specific Dictionaries

Properties that have an associated dictionary of file-format or metadata-format specific key-value pairs.

```
CFStringRef kCGImagePropertyTIFFDictionary;
CFStringRef kCGImagePropertyGIFDictionary;
CFStringRef kCGImagePropertyJFIFDictionary;
CFStringRef kCGImagePropertyExifDictionary;
CFStringRef kCGImagePropertyPNGDictionary;
CFStringRef kCGImagePropertyIPTCDictionary;
CFStringRef kCGImagePropertyGPSDictionary;
CFStringRef kCGImagePropertyRawDictionary;
CFStringRef kCGImagePropertyCIFFDictionary;
CFStringRef kCGImageProperty8BIMDictionary;
CFStringRef kCGImagePropertyDNGDictionary;
CFStringRef kCGImagePropertyExifAuxDictionary;
```

Constants

`kCGImagePropertyTIFFDictionary`

A dictionary of key-value pairs for an image that uses Tagged Image File Format (TIFF). See "TIFF Dictionary Keys" (page 2328).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGIFDictionary`

A dictionary of key-value pairs for an image that uses Graphics Interchange Format (GIF). See "GIF Dictionary Keys" (page 2316).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyJFIFDictionary`
A dictionary of key-value pairs for an image that uses JPEG File Interchange Format (JFIF). See "JFIF Dictionary Keys" (page 2326).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifDictionary`
A dictionary of key-value pairs for an image that uses Exchangeable Image File Format (EXIF). See "EXIF Dictionary Keys" (page 2307).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyPNGDictionary`
A dictionary of key-value pairs for an image that uses Portable Network Graphics (PNG) format. See "PNG Dictionary Keys" (page 2327).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCDictionary`
A dictionary of key-value pairs for an image that uses International Press Telecommunications Council (IPTC) metadata. See "IPTC Dictionary Keys" (page 2320).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDictionary`
A dictionary of key-value pairs for an image that has Global Positioning System (GPS) information. See "GPS Dictionary Keys" (page 2316).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyRawDictionary`
A dictionary of key-value pairs for an image that contains minimally processed, or raw, data.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFDictionary`
A dictionary of key-value pairs for an image that uses Camera Image File Format (CIFF). See "CIFF Dictionary Keys" (page 2332).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImageProperty8BIMDictionary`
A dictionary of key-value pairs for an Adobe Photoshop image. See "8BIM Dictionary Keys" (page 2332).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDNGDictionary`
A dictionary of key-value pairs for an image that uses the Digital Negative (DNG) archival format. See "DNG Dictionary Keys" (page 2331).

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxDictionary`
An auxiliary dictionary of key-value pairs for an image that uses Exchangeable Image File Format (EXIF).

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

Discussion
If any of these constants are returned by the functions`CGImageSourceCopyProperties` (page 240) or `CGImageSourceCopyPropertiesAtIndex` (page 241) the associated value is a dictionary of file-format or metadata-format specific key-value pairs.

Declared In
`CGImageProperties.h`

Camera Maker Dictionaries

Properties that have an associated dictionary of key-value pairs for a specific camera manufacturer.

```
CFStringRef kCGImagePropertyMakerCanonDictionary;
CFStringRef kCGImagePropertyMakerNikonDictionary;
CFStringRef kCGImagePropertyMakerMinoltaDictionary;
CFStringRef kCGImagePropertyMakerFujiDictionary;
CFStringRef kCGImagePropertyMakerOlympusDictionary;
CFStringRef kCGImagePropertyMakerPentaxDictionary;
```

Constants
`kCGImagePropertyMakerCanonDictionary`
A dictionary of key-value pairs for an image from a Canon camera. See "Canon Camera Dictionary Keys" (page 2337).

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonDictionary`
A dictionary of key-value pairs for an image from a Nikon camera. See "Nikon Camera Dictionary Keys" (page 2334).

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerMinoltaDictionary`
A dictionary of key-value pairs for an image from a Minolta camera.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerFujiDictionary`
A dictionary of key-value pairs for an image from a Fuji camera.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerOlympusDictionary`
A dictionary of key-value pairs for an image from a Olympus camera.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerPentaxDictionary`
A dictionary of key-value pairs for an image from a Pentax camera.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

Image Source Container Properties

Properties that apply to the container in general but not necessarily to any individual image in the container.

```
CFStringRef kCGImagePropertyFileSize;
```

Constants
`kCGImagePropertyFileSize`
The size of the image file in bytes, if known. If present, this key is a `CFNumber` value.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

Discussion
These properties can be returned by the function `CGImageSourceCopyProperties` (page 240).

Declared In
`CGImageProperties.h`

Individual Image Properties

Properties that apply to an individual image in an image source.

```
CFStringRef kCGImagePropertyDPIHeight;
CFStringRef kCGImagePropertyDPIWidth;
CFStringRef kCGImagePropertyPixelWidth;
CFStringRef kCGImagePropertyPixelHeight;
CFStringRef kCGImagePropertyDepth;
CFStringRef kCGImagePropertyOrientation;
CFStringRef kCGImagePropertyIsFloat;
CFStringRef kCGImagePropertyIsIndexed;
CFStringRef kCGImagePropertyHasAlpha;
CFStringRef kCGImagePropertyColorModel;
CFStringRef kCGImagePropertyProfileName;
```

Constants
`kCGImagePropertyDPIHeight`
The resolution, in dots per inch, in the x dimension. If present, this key is a `CFNumber` value.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDPIWidth`
The resolution, in dots per inch, in the y dimension. If present, this key is a `CFNumber` value.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyPixelWidth`
The number of pixels in the x dimension. If present, this key is a `CFNumber` value.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyPixelHeight`
The number of pixels in the y dimension. If present, this key is a `CFNumber` value.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDepth`
The number of bits in each color sample of each pixel. If present, this key is a `CFNumber` value.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyOrientation`
The intended display orientation of the image. If present, this key is a `CFNumber` value with the same value as defined by the TIFF and EXIF specifications. The value specifies where the origin `(0,0)` of the image is locates, as shown in Table 49-1. If not present, a value of 1 is assumed.

Table 49-1

Value	Location of the origin of the image
1	Top, left
2	Top, right
3	Bottom, right
4	Bottom, left
5	Left, top
6	Right, top
7	Right, bottom
8	Left, bottom

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIsFloat`
Whether or not the image contains floating-point pixel samples. The value of this key is `kCFBooleanTrue` if the image contains them.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIsIndexed`
Whether or not the image contains indexed pixel samples (sometimes called paletted samples). The value of this key is `kCFBooleanTrue` if the image contains them.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyHasAlpha`
Whether or not the image has an alpha channel. The value of this key is `kCFBooleanTrue` if the image contains an alpha channel.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyColorModel`
The color model of the image such as, "RGB", "CMYK", "Gray", or "Lab". The value of this key is CFStringRef.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyProfileName`
The name of the optional ICC profile embedded in the image, if known. If present, the value of this key is a CFStringRef.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

Discussion
These properties can be returned by the function `CGImageSourceCopyPropertiesAtIndex` (page 241).

Declared In
`CGImageProperties.h`

Color Model Values

Values for the color model property.

```
const CFStringRef kCGImagePropertyColorModelRGB;
const CFStringRef kCGImagePropertyColorModelGray;
const CFStringRef kCGImagePropertyColorModelCMYK;
const CFStringRef kCGImagePropertyColorModelLab;
```

Constants
`kCGImagePropertyColorModelRGB`
An RGB color model.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyColorModelGray`
A Gray color model.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyColorModelCMYK`
A CMYK color model.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyColorModelLab`

A Lab color model.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

Discussion
A color model describes how color values are represented mathematically. A color space is a color model combined with a definition of how to interpret values within the model.

Declared In
`CGImageProperties.h`

EXIF Dictionary Keys

Keys for for an image that uses Exchangeable Image File Format (EXIF).

```
const CFStringRef kCGImagePropertyExifExposureTime;
const CFStringRef kCGImagePropertyExifFNumber;
const CFStringRef kCGImagePropertyExifExposureProgram;
const CFStringRef kCGImagePropertyExifSpectralSensitivity;
const CFStringRef kCGImagePropertyExifISOSpeedRatings;
const CFStringRef kCGImagePropertyExifOECF;
const CFStringRef kCGImagePropertyExifVersion;
const CFStringRef kCGImagePropertyExifDateTimeOriginal;
const CFStringRef kCGImagePropertyExifDateTimeDigitized;
const CFStringRef kCGImagePropertyExifComponentsConfiguration;
const CFStringRef kCGImagePropertyExifCompressedBitsPerPixel;
const CFStringRef kCGImagePropertyExifShutterSpeedValue;
const CFStringRef kCGImagePropertyExifApertureValue;
const CFStringRef kCGImagePropertyExifBrightnessValue;
const CFStringRef kCGImagePropertyExifExposureBiasValue;
const CFStringRef kCGImagePropertyExifMaxApertureValue;
const CFStringRef kCGImagePropertyExifSubjectDistance;
const CFStringRef kCGImagePropertyExifMeteringMode;
const CFStringRef kCGImagePropertyExifLightSource;
const CFStringRef kCGImagePropertyExifFlash;
const CFStringRef kCGImagePropertyExifFocalLength;
const CFStringRef kCGImagePropertyExifSubjectArea;
const CFStringRef kCGImagePropertyExifMakerNote;
const CFStringRef kCGImagePropertyExifUserComment;
const CFStringRef kCGImagePropertyExifSubsecTime;
const CFStringRef kCGImagePropertyExifSubsecTimeOrginal;
const CFStringRef kCGImagePropertyExifSubsecTimeDigitized;
const CFStringRef kCGImagePropertyExifFlashPixVersion;
const CFStringRef kCGImagePropertyExifColorSpace;
const CFStringRef kCGImagePropertyExifPixelXDimension;
const CFStringRef kCGImagePropertyExifPixelYDimension;
const CFStringRef kCGImagePropertyExifRelatedSoundFile;
const CFStringRef kCGImagePropertyExifFlashEnergy;
const CFStringRef kCGImagePropertyExifSpatialFrequencyResponse;
const CFStringRef kCGImagePropertyExifFocalPlaneXResolution;
const CFStringRef kCGImagePropertyExifFocalPlaneYResolution;
const CFStringRef kCGImagePropertyExifFocalPlaneResolutionUnit;
const CFStringRef kCGImagePropertyExifSubjectLocation;
const CFStringRef kCGImagePropertyExifExposureIndex;
const CFStringRef kCGImagePropertyExifSensingMethod;
const CFStringRef kCGImagePropertyExifFileSource;
const CFStringRef kCGImagePropertyExifSceneType;
const CFStringRef kCGImagePropertyExifCFAPattern;
const CFStringRef kCGImagePropertyExifCustomRendered;
const CFStringRef kCGImagePropertyExifExposureMode;
const CFStringRef kCGImagePropertyExifWhiteBalance;
const CFStringRef kCGImagePropertyExifDigitalZoomRatio;
const CFStringRef kCGImagePropertyExifFocalLenIn35mmFilm;
const CFStringRef kCGImagePropertyExifSceneCaptureType;
const CFStringRef kCGImagePropertyExifGainControl;
const CFStringRef kCGImagePropertyExifContrast;
const CFStringRef kCGImagePropertyExifSaturation;
const CFStringRef kCGImagePropertyExifSharpness;
const CFStringRef kCGImagePropertyExifDeviceSettingDescription;
const CFStringRef kCGImagePropertyExifSubjectDistRange;
const CFStringRef kCGImagePropertyExifImageUniqueID;
const CFStringRef kCGImagePropertyExifGamma;
```

Constants

`kCGImagePropertyExifExposureTime`
The exposure time.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFNumber`
The F number.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifExposureProgram`
The exposure program.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSpectralSensitivity`
The spectral sensitivity of each channel.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifISOSpeedRatings`
ISO speed ratings.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifOECF`
The opto-electrical conversion function (OECF), which defines the relationship between the optical input of the camera and the image values.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifVersion`
The version.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifDateTimeOriginal`
The original date and time.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifDateTimeDigitized`
The digitized date and time.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifComponentsConfiguration`
The components configuration. For compressed data, specifies that the channels of each component are arranged in increasing numeric order (from first component to the fourth).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifCompressedBitsPerPixel`
The compressed bits per pixel.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifShutterSpeedValue`
The shutter speed value.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifApertureValue`
The aperture value.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifBrightnessValue`
The brightness value.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifExposureBiasValue`
The exposure bias value.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifMaxApertureValue`
The maximum aperture value.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSubjectDistance`
The distance to the subject, in meters.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifMeteringMode`
The metering mode.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifLightSource`
The light source.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFlash`
The flash status when the image was shot.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFocalLength`
The focal length.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSubjectArea`
The subject area.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifMakerNote`
A maker note.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifUserComment`
A user comment.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSubsecTime`
The fraction of seconds for the date and time tag.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSubsecTimeOrginal`
The fraction of seconds for the original date and time tag.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSubsecTimeDigitized`
The fraction of seconds for the digitized time tag.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFlashPixVersion`
The FlashPix version supported by an FPXR file. FlashPix is a format for multi-resolution, tiled images, that facilitates fast onscreen viewing.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifColorSpace`
The color space.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifPixelXDimension`
The pixel x dimension.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifPixelYDimension`
The pixel y dimension.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifRelatedSoundFile`
A related sound file.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFlashEnergy`
The strobe energy when the image was captures, in beam candle power seconds.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSpatialFrequencyResponse`
The spatial frequency table and spatial frequency response values in the direction of image width, image height, and diagonal directions. See ISO 12233..

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFocalPlaneXResolution`
The number of image-width pixels (x) per focal plane resolution unit.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFocalPlaneYResolution`
The number of image-height pixels (y)per focal plane resolution unit.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFocalPlaneResolutionUnit`
The unit of measurement for the focal plane x and y tags.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSubjectLocation`
The location of the scene's primary subject.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifExposureIndex`
The selected exposure index.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSensingMethod`
The sensor type of the camera or input device.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFileSource`
The image source.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSceneType`
The scene type.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifCFAPattern`
The color filter array (CFA) pattern, which is the geometric patter of the image sensor for a 1-chip color sensor area.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifCustomRendered`
Special rendering performed on the image data.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifExposureMode`
The exposure mode setting.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifWhiteBalance`
The white balance mode.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifDigitalZoomRatio`
The digital zoom ratio.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifFocalLenIn35mmFilm`
The equivalent focal length in 35 mm film.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSceneCaptureType`
The scene capture type (standard, landscape, portrait, night).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifGainControl`
The gain adjustment applied to the image.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyExifContrast`
The contrast applied to the image.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSaturation`
The saturation applied to the image.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSharpness`
The sharpness applied to the image.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifDeviceSettingDescription`
For a particular camera mode, indicates the conditions for taking the picture.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifSubjectDistRange`
The subject distance range.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifImageUniqueID`
The unique ID of the image.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifGamma`
The gamma setting.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

EXIF Auxiliary Dictionary Keys

Auxiliary keys for for an image that uses Exchangeable Image File Format (EXIF).

```
const CFStringRef kCGImagePropertyExifAuxLensInfo;
const CFStringRef kCGImagePropertyExifAuxLensModel;
const CFStringRef kCGImagePropertyExifAuxSerialNumber;
const CFStringRef kCGImagePropertyExifAuxLensID;
const CFStringRef kCGImagePropertyExifAuxLensSerialNumber;
const CFStringRef kCGImagePropertyExifAuxImageNumber;
const CFStringRef kCGImagePropertyExifAuxFlashCompensation;
const CFStringRef kCGImagePropertyExifAuxOwnerName;
const CFStringRef kCGImagePropertyExifAuxFirmware;
```

Constants

`kCGImagePropertyExifAuxLensInfo`
Lens information.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxLensModel`
The lens model.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxSerialNumber`
The serial number.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxLensID`
The lens ID.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxLensSerialNumber`
The lens serial number.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxImageNumber`
The image number.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxFlashCompensation`
Flash compensation.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxOwnerName`
The owner name.
Available in Mac OS X v10.5 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyExifAuxFirmware`
Firmware information.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

GIF Dictionary Keys

Keys for an image that uses Graphics Interchange Format (GIF).

```
const CFStringRef kCGImagePropertyGIFLoopCount;
const CFStringRef kCGImagePropertyGIFDelayTime;
const CFStringRef kCGImagePropertyGIFImageColorMap;
const CFStringRef kCGImagePropertyGIFHasGlobalColorMap;
```

Constants
`kCGImagePropertyGIFLoopCount`
The loop count.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGIFDelayTime`
The delay time.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGIFImageColorMap`
The image color map.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGIFHasGlobalColorMap`
Whether or not the GIF has a global color map.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

GPS Dictionary Keys

Keys for an image that has Global Positioning System (GPS) information.

```
const CFStringRef kCGImagePropertyGPSVersion;
const CFStringRef kCGImagePropertyGPSLatitudeRef;
const CFStringRef kCGImagePropertyGPSLatitude;
const CFStringRef kCGImagePropertyGPSLongitudeRef;
const CFStringRef kCGImagePropertyGPSLongitude;
const CFStringRef kCGImagePropertyGPSAltitudeRef;
const CFStringRef kCGImagePropertyGPSAltitude;
const CFStringRef kCGImagePropertyGPSTimeStamp;
const CFStringRef kCGImagePropertyGPSSatellites;
const CFStringRef kCGImagePropertyGPSStatus;
const CFStringRef kCGImagePropertyGPSMeasureMode;
const CFStringRef kCGImagePropertyGPSDOP;
const CFStringRef kCGImagePropertyGPSSpeedRef;
const CFStringRef kCGImagePropertyGPSSpeed;
const CFStringRef kCGImagePropertyGPSTrackRef;
const CFStringRef kCGImagePropertyGPSTrack;
const CFStringRef kCGImagePropertyGPSImgDirectionRef;
const CFStringRef kCGImagePropertyGPSImgDirection;
const CFStringRef kCGImagePropertyGPSMapDatum;
const CFStringRef kCGImagePropertyGPSDestLatitudeRef;
const CFStringRef kCGImagePropertyGPSDestLatitude;
const CFStringRef kCGImagePropertyGPSDestLongitudeRef;
const CFStringRef kCGImagePropertyGPSDestLongitude;
const CFStringRef kCGImagePropertyGPSDestBearingRef;
const CFStringRef kCGImagePropertyGPSDestBearing;
const CFStringRef kCGImagePropertyGPSDestDistanceRef;
const CFStringRef kCGImagePropertyGPSDestDistance;
const CFStringRef kCGImagePropertyGPSProcessingMethod;
const CFStringRef kCGImagePropertyGPSAreaInformation;
const CFStringRef kCGImagePropertyGPSDateStamp;
const CFStringRef kCGImagePropertyGPSDifferental;
```

Constants

`kCGImagePropertyGPSVersion`

The version.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSLatitudeRef`

Whether the latitude is northern or southern.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSLatitude`

The latitude.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSLongitudeRef`

Whether the longitude is east or west.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSLongitude`
The longitude.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSAltitudeRef`
The reference altitude.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSAltitude`
The altitude.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSTimeStamp`
The time as UTC (Coordinated Universal Time).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSSatellites`
The satellites used for GPS measurements.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSStatus`
The status of the GPS receiver.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSMeasureMode`
The measurement mode.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDOP`
The data degree of precision (DOP).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSSpeedRef`
The unit for expressing the GPS receiver speed of movement.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSSpeed`
The GPS receiver speed of movement.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSTrackRef`
The reference for the direction of GPS receiver movement.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSTrack`
The direction of GPS receiver movement.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSImgDirectionRef`
The reference for the direction of the image.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSImgDirection`
The direction of the image.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSMapDatum`
The geodetic survey data used by the GPS receiver.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestLatitudeRef`
Whether the latitude of the destination point is northern or southern.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestLatitude`
The latitude of the destination point.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestLongitudeRef`
Whether the longitude of the destination point is east or west.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestLongitude`
The longitude of the destination point.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestBearingRef`
The reference for giving the bearing to the destination point.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestBearing`
The bearing to the destination point.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestDistanceRef`
The units for expressing the distance to the destination point.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDestDistance`
The distance to the destination point.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSProcessingMethod`
The name of the method used for finding a location.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSAreaInformation`
The name of the GPS area.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDateStamp`
The data and time information relative to Coordinated Universal Time (UTC).

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyGPSDifferental`
Whether differential correction is applied to the GPS receiver.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

IPTC Dictionary Keys

Keys for an image that uses International Press Telecommunications Council (IPTC) metadata.

```
const CFStringRef kCGImagePropertyIPTCObjectTypeReference;
const CFStringRef kCGImagePropertyIPTCObjectAttributeReference;
const CFStringRef kCGImagePropertyIPTCObjectName;
const CFStringRef kCGImagePropertyIPTCEditStatus;
const CFStringRef kCGImagePropertyIPTCEditorialUpdate;
const CFStringRef kCGImagePropertyIPTCUrgency;
const CFStringRef kCGImagePropertyIPTCSubjectReference;
const CFStringRef kCGImagePropertyIPTCCategory;
const CFStringRef kCGImagePropertyIPTCSupplementalCategory;
const CFStringRef kCGImagePropertyIPTCFixtureIdentifier;
const CFStringRef kCGImagePropertyIPTCKeywords;
const CFStringRef kCGImagePropertyIPTCContentLocationCode;
const CFStringRef kCGImagePropertyIPTCContentLocationName;
const CFStringRef kCGImagePropertyIPTCReleaseDate;
const CFStringRef kCGImagePropertyIPTCReleaseTime;
const CFStringRef kCGImagePropertyIPTCExpirationDate;
const CFStringRef kCGImagePropertyIPTCExpirationTime;
const CFStringRef kCGImagePropertyIPTCSpecialInstructions;
const CFStringRef kCGImagePropertyIPTCActionAdvised;
const CFStringRef kCGImagePropertyIPTCReferenceService;
const CFStringRef kCGImagePropertyIPTCReferenceDate;
const CFStringRef kCGImagePropertyIPTCReferenceNumber;
const CFStringRef kCGImagePropertyIPTCDateCreated;
const CFStringRef kCGImagePropertyIPTCTimeCreated;
const CFStringRef kCGImagePropertyIPTCDigitalCreationDate;
const CFStringRef kCGImagePropertyIPTCDigitalCreationTime;
const CFStringRef kCGImagePropertyIPTCOriginatingProgram;
const CFStringRef kCGImagePropertyIPTCProgramVersion;
const CFStringRef kCGImagePropertyIPTCObjectCycle;
const CFStringRef kCGImagePropertyIPTCByline;
const CFStringRef kCGImagePropertyIPTCBylineTitle;
const CFStringRef kCGImagePropertyIPTCCity;
const CFStringRef kCGImagePropertyIPTCSubLocation;
const CFStringRef kCGImagePropertyIPTCProvinceState;
const CFStringRef kCGImagePropertyIPTCCountryPrimaryLocationCode;
const CFStringRef kCGImagePropertyIPTCCountryPrimaryLocationName;
const CFStringRef kCGImagePropertyIPTCOriginalTransmissionReference;
const CFStringRef kCGImagePropertyIPTCHeadline;
const CFStringRef kCGImagePropertyIPTCCredit;
const CFStringRef kCGImagePropertyIPTCSource;
const CFStringRef kCGImagePropertyIPTCCopyrightNotice;
const CFStringRef kCGImagePropertyIPTCContact;
const CFStringRef kCGImagePropertyIPTCCaptionAbstract;
const CFStringRef kCGImagePropertyIPTCWriterEditor;
const CFStringRef kCGImagePropertyIPTCImageType;
const CFStringRef kCGImagePropertyIPTCImageOrientation;
const CFStringRef kCGImagePropertyIPTCLanguageIdentifier;
const CFStringRef kCGImagePropertyIPTCStarRating;
```

Constants

`kCGImagePropertyIPTCObjectTypeReference`

The object type.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCObjectAttributeReference`
The object attribute.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCObjectName`
The object name.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCEditStatus`
The edit status.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCEditorialUpdate`
An editorial update.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCUrgency`
The urgency level.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCSubjectReference`
The subject.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCCategory`
The category.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCSupplementalCategory`
A supplemental category.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCFixtureIdentifier`
A fixture identifier.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCKeywords`
Keywords.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCContentLocationCode`
The content location code.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCContentLocationName`
The content location name.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCReleaseDate`
The release date.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCReleaseTime`
The release time.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCExpirationDate`
The expiration date.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCExpirationTime`
The expiration time.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCSpecialInstructions`
Special instructions.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCActionAdvised`
The advised action.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCReferenceService`
The reference service.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCReferenceDate`
The reference date.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCReferenceNumber`
The reference number.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCDateCreated`
The date created.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCTimeCreated`
The time created.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCDigitalCreationDate`
The digital creation date.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCDigitalCreationTime`
The digital creation time.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCOriginatingProgram`
The originating program.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCProgramVersion`
The program version.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCObjectCycle`
The object cycle.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCByline`
The byline.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCBylineTitle`
The byline title.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCCity`
The city.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCSubLocation`
The sublocation.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCProvinceState`
The province or state.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCCountryPrimaryLocationCode`
The country primary location code.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCCountryPrimaryLocationName`
The country primary location name.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCOriginalTransmissionReference`
The original transmission reference.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCHeadline`
The headline.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCCredit`
Credit information.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCSource`
The source.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCCopyrightNotice`
The copyright notice.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCContact`
Contact information.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCCaptionAbstract`
The caption abstract.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCWriterEditor`
The writer or editor.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCImageType`
The image type.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCImageOrientation`
The image orientation.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCLanguageIdentifier`
The language identifier.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyIPTCStarRating`
The star rating.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

Discussion
IPTC constants are metadata elements of the Information Interchange Model (IIM) used to provide information about images. The IIM was developer by the Newspaper Association of America (NAA) and the International Press Telecommunications Council (IPTC).

Declared In
`CGImageProperties.h`

JFIF Dictionary Keys

Keys for an image that uses JPEG File Interchange Format (JFIF).

```
const CFStringRef kCGImagePropertyJFIFVersion;
const CFStringRef kCGImagePropertyJFIFXDensity;
const CFStringRef kCGImagePropertyJFIFYDensity;
const CFStringRef kCGImagePropertyJFIFDensityUnit;
const CFStringRef kCGImagePropertyJFIFIsProgressive;
```

Constants

`kCGImagePropertyJFIFVersion`
The version.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyJFIFXDensity`
The x density.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyJFIFYDensity`
The y density.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyJFIFDensityUnit`
The density unit.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyJFIFIsProgressive`
Whether or not the image is progressive.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

PNG Dictionary Keys

Keys for an image that uses Portable Network Graphics (PNG) format.

```
const CFStringRef kCGImagePropertyPNGGamma;
const CFStringRef kCGImagePropertyPNGInterlaceType;
const CFStringRef kCGImagePropertyPNGXPixelsPerMeter;
const CFStringRef kCGImagePropertyPNGYPixelsPerMeter;
const CFStringRef kCGImagePropertyPNGsRGBIntent;
const CFStringRef kCGImagePropertyPNGChromaticities;
```

Constants

`kCGImagePropertyPNGGamma`
The gamma value.
Available in Mac OS X v10.4 and later.
Declared in `CGImageProperties.h`.

`kCGImagePropertyPNGInterlaceType`
The interlace type.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyPNGXPixelsPerMeter`
The number of x pixels per meter.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyPNGYPixelsPerMeter`
The number of y pixels per meter.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyPNGsRGBIntent`
The sRGB intent.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyPNGChromaticities`
The chromaticities.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

TIFF Dictionary Keys

Keys for an image that uses Tagged Image File Format (TIFF).

```
const CFStringRef kCGImagePropertyTIFFCompression;
const CFStringRef kCGImagePropertyTIFFPhotometricInterpretation;
const CFStringRef kCGImagePropertyTIFFDocumentName;
const CFStringRef kCGImagePropertyTIFFImageDescription;
const CFStringRef kCGImagePropertyTIFFMake;
const CFStringRef kCGImagePropertyTIFFModel;
const CFStringRef kCGImagePropertyTIFFOrientation;
const CFStringRef kCGImagePropertyTIFFXResolution;
const CFStringRef kCGImagePropertyTIFFYResolution;
const CFStringRef kCGImagePropertyTIFFResolutionUnit;
const CFStringRef kCGImagePropertyTIFFSoftware;
const CFStringRef kCGImagePropertyTIFFTransferFunction;
const CFStringRef kCGImagePropertyTIFFDateTime;
const CFStringRef kCGImagePropertyTIFFArtist;
const CFStringRef kCGImagePropertyTIFFHostComputer;
const CFStringRef kCGImagePropertyTIFFCopyright;
const CFStringRef kCGImagePropertyTIFFWhitePoint;
const CFStringRef kCGImagePropertyTIFFPrimaryChromaticities;
```

Constants

`kCGImagePropertyTIFFCompression`

The compression scheme used on the image data.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFPhotometricInterpretation`

The color space of the image data.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFDocumentName`

The document name.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFImageDescription`

The image description.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFMake`

The camera or input device make.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFModel`

A camera or input device model.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFOrientation`

The image orientation.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFXResolution`
The number of pixels per resolution unit in the image width direction.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFYResolution`
The number of pixels per resolution unit in the image height direction.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFResolutionUnit`
The units of resolution.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFSoftware`
The name and version of the software used for image creation.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFTransferFunction`
The transfer function, in tabular format, used to map pixel components from a nonlinear form into a linear form.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFDateTime`
The date and time.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFArtist`
The artist.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFHostComputer`
The computer or operation system used when the image was created.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFCopyright`
Copyright information.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFWhitePoint`
The white point.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyTIFFPrimaryChromaticities`
The chromaticities of the primaries of the image.

Available in Mac OS X v10.4 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

DNG Dictionary Keys

Keys for an image that uses the Digital Negative (DNG) archival format.

```
CFStringRef  kCGImagePropertyDNGVersion;
CFStringRef  kCGImagePropertyDNGBackwardVersion;
CFStringRef  kCGImagePropertyDNGUniqueCameraModel;
CFStringRef kCGImagePropertyDNGLocalizedCameraModel;
CFStringRef  kCGImagePropertyDNGCameraSerialNumber;
CFStringRef  kCGImagePropertyDNGLensInfo;
```

Constants
`kCGImagePropertyDNGVersion`
An encoding of the four-tier version number.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDNGBackwardVersion`
The oldest version for which a file is compatible.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDNGUniqueCameraModel`
A unique, nonlocalized name for the camera mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDNGLocalizedCameraModel`
The localized camera model name.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDNGCameraSerialNumber`
The camera serial number.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyDNGLensInfo`
Information about the lens used for the image.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

8BIM Dictionary Keys

A key for an Adobe Photoshop image.

```
CFStringRef  kCGImageProperty8BIMLayerNames;
```

Constants

`kCGImageProperty8BIMLayerNames`

The layer names for an Adobe Photoshop file.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

Declared In

`CGImageProperties.h`

CIFF Dictionary Keys

Keys for an image that uses Camera Image File Format (CIFF).

```
CFStringRef  kCGImagePropertyCIFFDescription;
CFStringRef  kCGImagePropertyCIFFFirmware;
CFStringRef  kCGImagePropertyCIFFOwnerName;
CFStringRef  kCGImagePropertyCIFFImageName;
CFStringRef  kCGImagePropertyCIFFImageFileName;
CFStringRef  kCGImagePropertyCIFFReleaseMethod;
CFStringRef  kCGImagePropertyCIFFReleaseTiming;
CFStringRef  kCGImagePropertyCIFFRecordID;
CFStringRef  kCGImagePropertyCIFFSelfTimingTime;
CFStringRef  kCGImagePropertyCIFFCameraSerialNumber;
CFStringRef  kCGImagePropertyCIFFImageSerialNumber;
CFStringRef  kCGImagePropertyCIFFContinuousDrive;
CFStringRef  kCGImagePropertyCIFFFocusMode;
CFStringRef  kCGImagePropertyCIFFMeteringMode;
CFStringRef  kCGImagePropertyCIFFShootingMode;
CFStringRef  kCGImagePropertyCIFFLensMaxMM;
CFStringRef  kCGImagePropertyCIFFLensMinMM;
CFStringRef  kCGImagePropertyCIFFLensModel;
CFStringRef  kCGImagePropertyCIFFWhiteBalanceIndex;
CFStringRef  kCGImagePropertyCIFFFlashExposureComp;
CFStringRef  kCGImagePropertyCIFFMeasuredEV;
```

Constants

`kCGImagePropertyCIFFDescription`

The camera description..

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFFirmware`

The firmware version.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFOwnerName`
The owner name.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFImageName`
The image name.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFImageFileName`
The image file name.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFReleaseMethod`
The release method.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFReleaseTiming`
The release timing.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFRecordID`
The record ID>

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFSelfTimingTime`
The self timing time.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFCameraSerialNumber`
The camera serial number.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFImageSerialNumber`
The image serial number.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFContinuousDrive`
The continuous drive mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFFocusMode`
The focus mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFMeteringMode`
The metering mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFShootingMode`
The shooting mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFLensMaxMM`
The maximum lens length.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFLensMinMM`
The minimum lens length.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFLensModel`
The lens model.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFWhiteBalanceIndex`
The white balance index.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFFlashExposureComp`
The flash exposure compensation.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyCIFFMeasuredEV`
The measured EV.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

Nikon Camera Dictionary Keys

Keys for an image from a Nikon camera.

```
CFStringRef  kCGImagePropertyMakerNikonISOSetting;
CFStringRef  kCGImagePropertyMakerNikonColorMode;
CFStringRef  kCGImagePropertyMakerNikonQuality;
CFStringRef  kCGImagePropertyMakerNikonWhiteBalanceMode;
CFStringRef  kCGImagePropertyMakerNikonSharpenMode;
CFStringRef  kCGImagePropertyMakerNikonFocusMode;
CFStringRef  kCGImagePropertyMakerNikonFlashSetting;
CFStringRef  kCGImagePropertyMakerNikonISOSelection;
CFStringRef  kCGImagePropertyMakerNikonFlashExposureComp;
CFStringRef  kCGImagePropertyMakerNikonImageAdjustment;
CFStringRef  kCGImagePropertyMakerNikonLensAdapter;
CFStringRef  kCGImagePropertyMakerNikonLensType;
CFStringRef  kCGImagePropertyMakerNikonLensInfo;
CFStringRef  kCGImagePropertyMakerNikonFocusDistance;
CFStringRef  kCGImagePropertyMakerNikonDigitalZoom;
CFStringRef  kCGImagePropertyMakerNikonShootingMode;
CFStringRef  kCGImagePropertyMakerNikonShutterCount;
CFStringRef  kCGImagePropertyMakerNikonCameraSerialNumber;
```

Constants

`kCGImagePropertyMakerNikonISOSetting`

The ISO setting.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonColorMode`

The color mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonQuality`

The quality setting.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonWhiteBalanceMode`

The white balance mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonSharpenMode`

The sharpening mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonFocusMode`

The focus mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonFlashSetting`

The flash setting.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonISOSelection`
The ISO selection.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonFlashExposureComp`
The flash exposure compensation.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonImageAdjustment`
Image adjustment setting.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonLensAdapter`
The lens adapter.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonLensType`
The lens type.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonLensInfo`
Lens information.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonFocusDistance`
The focus distance.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonDigitalZoom`
The digital zoom setting.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonShootingMode`
The shooting mode.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonShutterCount`
The shutter count.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerNikonCameraSerialNumber`
The camera serial number.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

Canon Camera Dictionary Keys

Keys for an image from a Canon camera.

```
CFStringRef  kCGImagePropertyMakerCanonOwnerName;
CFStringRef  kCGImagePropertyMakerCanonCameraSerialNumber;
CFStringRef  kCGImagePropertyMakerCanonImageSerialNumber;
CFStringRef  kCGImagePropertyMakerCanonFlashExposureComp;
CFStringRef  kCGImagePropertyMakerCanonContinuousDrive;
CFStringRef  kCGImagePropertyMakerCanonLensModel;
CFStringRef  kCGImagePropertyMakerCanonFirmware;
CFStringRef  kCGImagePropertyMakerCanonAspectRatioInfo;
```

Constants
`kCGImagePropertyMakerCanonOwnerName`
The owner name.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerCanonCameraSerialNumber`
The camera serial number.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerCanonImageSerialNumber`
The image serial number.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerCanonFlashExposureComp`
The flash exposure compensation.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerCanonContinuousDrive`
The presence of a continuous drive.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerCanonLensModel`
The lens model.

Available in Mac OS X v10.5 and later.

Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerCanonFirmware`
: The firmware version.

  Available in Mac OS X v10.5 and later.

  Declared in `CGImageProperties.h`.

`kCGImagePropertyMakerCanonAspectRatioInfo`
: The image aspect ratio.

  Available in Mac OS X v10.5 and later.

  Declared in `CGImageProperties.h`.

Declared In
`CGImageProperties.h`

CGAffineTransform Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGAffineTransform.h
Companion guide	Quartz 2D Programming Guide

Overview

The `CGAffineTransform` data structure represents a matrix used for affine transformations. A transformation specifies how points in one coordinate system map to points in another coordinate system. An affine transformation is a special type of mapping that preserves parallel lines in a path but does not necessarily preserve lengths or angles. Scaling, rotation, and translation are the most commonly used manipulations supported by affine transforms, but skewing is also possible.

Quartz provides functions that create, concatenate, and apply affine transformations using the `CGAffineTransform` data structure. For information on how to use affine transformation functions, see *Quartz 2D Programming Guide*.

You typically do not need to create an affine transform directly—*CGContext Reference* describes functions that modify the current affine transform. If you don't plan to reuse an affine transform, you may want to use `CGContextScaleCTM` (page 105), `CGContextRotateCTM` (page 104), `CGContextTranslateCTM` (page 136), or `CGContextConcatCTM` (page 82).

Functions by Task

Creating an Affine Transformation Matrix

`CGAffineTransformMake` (page 2342)
Returns an affine transformation matrix constructed from values you provide.

`CGAffineTransformMakeRotation` (page 2344)
Returns an affine transformation matrix constructed from a rotation value you provide.

`CGAffineTransformMakeScale` (page 2344)
Returns an affine transformation matrix constructed from scaling values you provide.

`CGAffineTransformMakeTranslation` (page 2345)
Returns an affine transformation matrix constructed from translation values you provide.

Modifying Affine Transformations

`CGAffineTransformTranslate` (page 2348)
Returns an affine transformation matrix constructed by translating an existing affine transform.

`CGAffineTransformScale` (page 2347)
Returns an affine transformation matrix constructed by scaling an existing affine transform.

`CGAffineTransformRotate` (page 2346)
Returns an affine transformation matrix constructed by rotating an existing affine transform.

`CGAffineTransformInvert` (page 2341)
Returns an affine transformation matrix constructed by inverting an existing affine transform.

`CGAffineTransformConcat` (page 2340)
Returns an affine transformation matrix constructed by combining two existing affine transforms.

Applying Affine Transformations

`CGPointApplyAffineTransform` (page 2348)
Returns the point resulting from an affine transformation of an existing point.

`CGSizeApplyAffineTransform` (page 2349)
Returns the height and width resulting from a transformation of an existing height and width.

`CGRectApplyAffineTransform` (page 2349)
Applies an affine transform to a rectangle.

Evaluating Affine Transforms

`CGAffineTransformIsIdentity` (page 2342)
Checks whether an affine transform is the identity transform.

`CGAffineTransformEqualToTransform` (page 2341)
Checks whether two affine transforms are equal.

Functions

CGAffineTransformConcat

Returns an affine transformation matrix constructed by combining two existing affine transforms.

```
CGAffineTransform CGAffineTransformConcat (
   CGAffineTransform t1,
   CGAffineTransform t2
);
```

Parameters

t1
The first affine transform.

t2

The second affine transform. This affine transform is concatenated to the first affine transform.

Return Value
A new affine transformation matrix. That is, t' = t1*t2.

Discussion
Concatenation combines two affine transformation matrices by multiplying them together. You might perform several concatenations in order to create a single affine transform that contains the cumulative effects of several transformations.

Note that matrix operations are not commutative—the order in which you concatenate matrices is important. That is, the result of multiplying matrix `t1` by matrix `t2` does not necessarily equal the result of multiplying matrix `t2` by matrix `t1`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

CGAffineTransformEqualToTransform

Checks whether two affine transforms are equal.

```
bool CGAffineTransformEqualToTransform (
   CGAffineTransform t1,
   CGAffineTransform t2
);
```

Parameters

t1

An affine transform.

t2

An affine transform.

Return Value
Returns `true` if `t1` and `t2` are equal, `false` otherwise.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGAffineTransform.h`

CGAffineTransformInvert

Returns an affine transformation matrix constructed by inverting an existing affine transform.

```
CGAffineTransform CGAffineTransformInvert (
   CGAffineTransform t
);
```

Parameters

t

An existing affine transform.

Return Value
A new affine transformation matrix. If the affine transform passed in parameter `t` cannot be inverted, Quartz returns the affine transform unchanged.

Discussion
Inversion is generally used to provide reverse transformation of points within transformed objects. Given the coordinates (x,y), which have been transformed by a given matrix to new coordinates (x′,y′), transforming the coordinates (x′,y′) by the inverse matrix produces the original coordinates (x,y).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

CGAffineTransformIsIdentity

Checks whether an affine transform is the identity transform.

```
bool CGAffineTransformIsIdentity (
   CGAffineTransform t
);
```

Parameters

t

The affine transform to check.

Return Value
Returns `true` if `t` is the identity transform, `false` otherwise.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGAffineTransform.h`

CGAffineTransformMake

Returns an affine transformation matrix constructed from values you provide.

```
CGAffineTransform CGAffineTransformMake (
   CGFloat a,
   CGFloat b,
   CGFloat c,
   CGFloat d,
   CGFloat tx,
   CGFloat ty
);
```

Parameters

a
The value at position [1,1] in the matrix.

b
The value at position [1,2] in the matrix.

c
The value at position [2,1] in the matrix.

d
The value at position [2,2] in the matrix.

tx
The value at position [3,1] in the matrix.

ty
The value at position [3,2] in the matrix.

Return Value
A new affine transform matrix constructed from the values you specify.

Discussion
This function creates a `CGAffineTransform` structure that represents a new affine transformation matrix, which you can use (and reuse, if you want) to transform a coordinate system. The matrix takes the following form:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Because the third column is always `(0,0,1)`, the `CGAffineTransform` data structure returned by this function contains values for only the first two columns.

If you want only to transform an object to be drawn, it is not necessary to construct an affine transform to do so. The most direct way to transform your drawing is by calling the appropriate `CGContext` function to adjust the current transformation matrix.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGAffineTransform.h`

CGAffineTransformMakeRotation

Returns an affine transformation matrix constructed from a rotation value you provide.

```
CGAffineTransform CGAffineTransformMakeRotation (
   CGFloat angle
);
```

Parameters

angle

The angle, in radians, by which this matrix rotates the coordinate system axes. A positive value specifies clockwise rotation, a negative value specifies counterclockwise.

Return Value

A new affine transformation matrix.

Discussion

This function creates a `CGAffineTransform` structure, which you can use (and reuse, if you want) to rotate a coordinate system. The matrix takes the following form:

$$\begin{bmatrix} \cos a & \sin a & 0 \\ -\sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Because the third column is always `(0,0,1)`, the `CGAffineTransform` data structure returned by this function contains values for only the first two columns.

These are the resulting equations that Quartz uses to apply the rotation to a point (x, y):

$$x' = x\cos a - y\sin a$$
$$y' = x\sin a + y\cos a$$

If you want only to rotate an object to be drawn, it is not necessary to construct an affine transform to do so. The most direct way to rotate your drawing is by calling the function `CGContextRotateCTM` (page 104).

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGAffineTransform.h`

CGAffineTransformMakeScale

Returns an affine transformation matrix constructed from scaling values you provide.

```
CGAffineTransform CGAffineTransformMakeScale (
   CGFloat sx,
   CGFloat sy
);
```

Parameters

sx

The factor by which to scale the x-axis of the coordinate system.

sy

The factor by which to scale the y-axis of the coordinate system.

Return Value
A new affine transformation matrix.

Discussion
This function creates a `CGAffineTransform` structure, which you can use (and reuse, if you want) to scale a coordinate system. The matrix takes the following form:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Because the third column is always `(0,0,1)`, the `CGAffineTransform` data structure returned by this function contains values for only the first two columns.

These are the resulting equations that Quartz uses to scale the coordinates of a point (x,y):

$$x' = x \cdot s_x$$
$$y' = y \cdot s_y$$

If you want only to scale an object to be drawn, it is not necessary to construct an affine transform to do so. The most direct way to scale your drawing is by calling the function `CGContextScaleCTM` (page 105).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

CGAffineTransformMakeTranslation

Returns an affine transformation matrix constructed from translation values you provide.

```
CGAffineTransform CGAffineTransformMake Translation (
   CGFloat tx,
   CGFloat ty
);
```

Parameters

tx

The value by which to move the x-axis of the coordinate system.

ty

The value by which to move the y-axis of the coordinate system.

Return Value
A new affine transform matrix.

Discussion
This function creates a `CGAffineTransform` structure. which you can use (and reuse, if you want) to move a coordinate system. The matrix takes the following form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Because the third column is always `(0,0,1)`, the `CGAffineTransform` data structure returned by this function contains values for only the first two columns.

These are the resulting equations Quartz uses to apply the translation to a point (x,y):

$$x' = x + t_x$$
$$y' = y + t_y$$

If you want only to move the location where an object is drawn, it is not necessary to construct an affine transform to do so. The most direct way to move your drawing is by calling the function `CGContextTranslateCTM` (page 136).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

CGAffineTransformRotate

Returns an affine transformation matrix constructed by rotating an existing affine transform.

```
CGAffineTransform CGAffineTransformRotate (
   CGAffineTransform t,
   CGFloat angle
);
```

Parameters

t

An existing affine transform.

angle

The angle, in radians, by which to rotate the affine transform.

Return Value
A new affine transformation matrix.

Discussion
You use this function to create a new affine transformation matrix by adding a rotation value to an existing affine transform. The resulting structure represents a new affine transform, which you can use (and reuse, if you want) to rotate a coordinate system.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

CGAffineTransformScale

Returns an affine transformation matrix constructed by scaling an existing affine transform.

```
CGAffineTransform CGAffineTransformScale (
   CGAffineTransform t,
   CGFloat sx,
   CGFloat sy
);
```

Parameters

t

An existing affine transform.

sx

The value by which to scale x values of the affine transform.

sy

The value by which to scale y values of the affine transform.

Return Value
A new affine transformation matrix.

Discussion
You use this function to create a new affine transformation matrix by adding scaling values to an existing affine transform. The resulting structure represents a new affine transform, which you can use (and reuse, if you want) to scale a coordinate system.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
HID Calibrator

Declared In
`CGAffineTransform.h`

CGAffineTransformTranslate

Returns an affine transformation matrix constructed by translating an existing affine transform.

```
CGAffineTransform CGAffineTransformTranslate (
   CGAffineTransform t,
   CGFloat tx,
   CGFloat ty
);
```

Parameters

t
: An existing affine transform.

tx
: The value by which to move x values with the affine transform.

ty
: The value by which to move y values with the affine transform.

Return Value
A new affine transformation matrix.

Discussion
You use this function to create a new affine transform by adding translation values to an existing affine transform. The resulting structure represents a new affine transform, which you can use (and reuse, if you want) to move a coordinate system.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

CGPointApplyAffineTransform

Returns the point resulting from an affine transformation of an existing point.

```
CGPoint CGPointApplyAffineTransform (
   CGPoint point,
   CGAffineTransform t
);
```

Parameters

point
: A point that specifies the x- and y-coordinates to transform.

t
: The affine transform to apply.

Return Value
A new point resulting from applying the specified affine transform to the existing point.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

CGRectApplyAffineTransform

Applies an affine transform to a rectangle.

```
CGRect CGRectApplyAffineTransform (
   CGRect rect,
   CGAffineTransform t
);
```

Parameters

rect

The rectangle whose corner points you want to transform.

t

The affine transform to apply to the `rect` parameter.

Return Value
The transformed rectangle.

Discussion
Because affine transforms do not preserve rectangles in general, the function `CGRectApplyAffineTransform` returns the smallest rectangle that contains the transformed corner points of the `rect` parameter. If the affine transform `t` consists solely of scaling and translation operations, then the returned rectangle coincides with the rectangle constructed from the four transformed corners.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CGAffineTransform.h`

CGSizeApplyAffineTransform

Returns the height and width resulting from a transformation of an existing height and width.

```
CGSize CGSizeApplyAffineTransform (
   CGSize size,
   CGAffineTransform t
);
```

Parameters

size

A size that specifies the height and width to transform.

t

The affine transform to apply.

Return Value
A new size resulting from applying the specified affine transform to the existing size.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGAffineTransform.h`

Data Types

CGAffineTransform

A structure for holding an affine transformation matrix.

```
struct CGAffineTransform {
    CGFloat a;
    CGFloat b;
    CGFloat c;
    CGFloat d;
    CGFloat tx;
    CGFloat ty;
};
typedef struct CGAffineTransform CGAffineTransform;
```

Fields

`a`
The entry at position [1,1] in the matrix.

`b`
The entry at position [1,2] in the matrix.

`c`
The entry at position [2,1] in the matrix.

`d`
The entry at position [2,2] in the matrix.

`tx`
The entry at position [3,1] in the matrix.

`ty`
The entry at position [3,2] in the matrix.

Discussion
In Quartz 2D, an affine transformation matrix is used to rotate, scale, translate, or skew the objects you draw in a graphics context. The `CGAffineTransform` type provides functions for creating, concatenating, and applying affine transformations.

In Quartz, affine transforms are represented by a 3 by 3 matrix:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Because the third column is always `(0,0,1)`, the `CGAffineTransform` data structure contains values for only the first two columns.

Conceptually, a Quartz affine transform multiplies a row vector representing each point (x,y) in your drawing by this matrix, producing a vector that represents the corresponding point (x′,y′):

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Given the 3 by 3 matrix, Quartz uses the following equations to transform a point (x, y) in one coordinate system into a resultant point (x′,y′) in another coordinate system.

$$x' = ax + cy + t_x$$
$$y' = bx + dy + t_y$$

The matrix thereby "links" two coordinate systems—it specifies how points in one coordinate system map to points in another.

Note that you do not typically need to create affine transforms directly. If you want only to draw an object that is scaled or rotated, for example, it is not necessary to construct an affine transform to do so. The most direct way to manipulate your drawing—whether by movement, scaling, or rotation—is to call the functions `CGContextTranslateCTM` (page 136), `CGContextScaleCTM` (page 105), or `CGContextRotateCTM` (page 104), respectively. You should generally only create an affine transform if you want to reuse it later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGAffineTransform.h`

Constants

CGAffineTransformIdentity

The identity transform.

```
const CGAffineTransform CGAffineTransformIdentity;
```

Constants

`CGAffineTransformIdentity`

The identity transform: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Available in Mac OS X v10.0 and later.

Declared in `CGAffineTransform.h`.

Declared In

`CGAffineTransform.h`

CGGeometry Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGGeometry.h
Companion guide	Quartz 2D Programming Guide

Overview

CGGeometry Reference defines structures for geometric primitives and functions that operate on them. The data structure `CGPoint` represents a point in a two-dimensional coordinate system. The data structure `CGRect` represents the location and dimensions of a rectangle. The data structure `CGSize` represents the dimensions of width and height.

Functions by Task

Creating a Geometric Primitive From a Dictionary Representation

`CGPointCreateDictionaryRepresentation` (page 2355)
Returns a dictionary representation of the provided point.

`CGSizeCreateDictionaryRepresentation` (page 2371)
Returns a dictionary representation of the provided size.

`CGRectCreateDictionaryRepresentation` (page 2358)
Returns a dictionary representation of the provided rectangle.

Creating a Dictionary Representation From a Geometric Primitive

`CGPointMakeWithDictionaryRepresentation` (page 2357)
Fills in a `CGPoint` structure using the contents of the provided dictionary.

`CGSizeMakeWithDictionaryRepresentation` (page 2372)
Fills in a `CGSize` structure using the contents of the provided dictionary.

`CGRectMakeWithDictionaryRepresentation` (page 2369)
Fills in a `CGRect` structure using the contents of the provided dictionary.

Creating a Geometric Primitive From Values

`CGPointMake` (page 2356)
Returns a `CGPoint` structure filled in with the coordinate values you provide.

`CGRectMake` (page 2368)
Returns a `CGRect` structure filled in with the coordinate and dimension values you provide.

`CGSizeMake` (page 2372)
Returns a `CGSize` structure filled in with dimension values you provide.

Modifying Rectangles

`CGRectDivide` (page 2359)
Divides a source rectangle into two component rectangles.

`CGRectInset` (page 2364)
Returns a rectangle that is smaller or larger than the source rectangle, with the same center point.

`CGRectIntegral` (page 2365)
Returns the smallest rectangle that results from converting the source rectangle values to integers.

`CGRectIntersection` (page 2365)
Returns the intersection of two rectangles.

`CGRectOffset` (page 2369)
Returns a rectangle with an origin that is offset from that of the source rectangle.

`CGRectStandardize` (page 2370)
Returns a rectangle with a positive width and height.

`CGRectUnion` (page 2371)
Returns the smallest rectangle that contains the two provided rectangles.

Comparing Values

`CGPointEqualToPoint` (page 2356)
Returns whether two points are equal.

`CGSizeEqualToSize` (page 2371)
Returns whether two sizes are equal.

`CGRectEqualToRect` (page 2359)
Returns whether two rectangles are equal in size and position.

`CGRectIntersectsRect` (page 2366)
Returns whether two rectangles intersect.

Checking for Membership

`CGRectContainsPoint` (page 2357)
Returns whether a rectangle contains a specified point.

`CGRectContainsRect` (page 2358)
Returns whether the first rectangle contains the second rectangle.

Getting Min, Mid, and Max Values

`CGRectGetMinX` (page 2362)
Returns the x-coordinate that establishes the left edge of a rectangle.

`CGRectGetMinY` (page 2363)
Returns the y-coordinate that establishes the bottom edge of a rectangle.

`CGRectGetMidX` (page 2361)
Returns the x- coordinate that establishes the center of a rectangle.

`CGRectGetMidY` (page 2362)
Returns the y-coordinate that establishes the center of a rectangle.

`CGRectGetMaxX` (page 2360)
Returns the x-coordinate that establishes the right edge of a rectangle.

`CGRectGetMaxY` (page 2361)
Returns the y-coordinate that establishes the top edge of a rectangle.

Getting Height and Width

`CGRectGetHeight` (page 2360)
Returns the height of a rectangle.

`CGRectGetWidth` (page 2363)
Returns the width of a rectangle.

Checking Rectangle Characteristics

`CGRectIsEmpty` (page 2366)
Returns whether a rectangle has zero width or height, or is a null rectangle.

`CGRectIsNull` (page 2368)
Returns whether a rectangle is invalid.

`CGRectIsInfinite` (page 2367)
Returns whether a rectangle is infinite.

`CGRectIsIntegral` (page 2367)
Returns whether the origin and size of the rectangle can be represented exactly as integers.

Functions

CGPointCreateDictionaryRepresentation

Returns a dictionary representation of the provided point.

```
CFDictionaryRef CGPointCreateDictionaryRepresentation(
    CGPoint point
);
```

Parameters

point

A point.

Return Value
The dictionary representation of the point.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGGeometry.h`

CGPointEqualToPoint

Returns whether two points are equal.

```
bool CGPointEqualToPoint (
   CGPoint point1,
   CGPoint point2
);
```

Parameters

point1

The first point to examine.

point2

The second point to examine.

Return Value
Returns `1` if the two specified points are the same; otherwise, `0`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGGeometry.h`

CGPointMake

Returns a `CGPoint` structure filled in with the coordinate values you provide.

```
CGPoint CGPointMake (
   CGFloat x,
   CGFloat y
);
```

Parameters

x

The x-coordinate of the point to construct.

y
The y-coordinate of the point to construct.

Return Value
Returns a `CGPoint` structure, representing a single (x,y) coordinate pair.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CALayerEssentials
CarbonSketch

Declared In
`CGGeometry.h`

CGPointMakeWithDictionaryRepresentation

Fills in a `CGPoint` structure using the contents of the provided dictionary.

```
bool CGPointMakeWithDictionaryRepresentation(
    CFDictionaryRef dict,
    CGPoint *point
);
```

Parameters
dict
A dictionary that was previously returned from the function `CGPointCreateDictionaryRepresentation` (page 2355).

point
On return, the point created from the provided dictionary.

Return Value
`true` if successful; `false` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGGeometry.h`

CGRectContainsPoint

Returns whether a rectangle contains a specified point.

```
bool CGRectContainsPoint (
   CGRect rect,
   CGPoint point
);
```

Parameters
rect
The rectangle to examine.

point
The point to examine.

Return Value
Returns `1` if the specified point is located within the specified rectangle; otherwise, `0`.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGGeometry.h`

CGRectContainsRect

Returns whether the first rectangle contains the second rectangle.

```
bool CGRectContainsRect (
   CGRect rect1,
   CGRect rect2
);
```

Parameters
rect1
The rectangle to examine for containment of the rectangle passed in `rect2`.

rect2
The rectangle to examine for being contained in the rectangle passed in `rect1`.

Return Value
Returns `1` if the rectangle specified by `rect2` is contained in the rectangle passed in `rect1`; otherwise, `0`. The first rectangle contains the second if the union of the two rectangles is equal to the first rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGGeometry.h`

CGRectCreateDictionaryRepresentation

Returns a dictionary representation of the provided rectangle.

```
CFDictionaryRef CGRectCreateDictionaryRepresentation(
    CGRect rect
);
```

Parameters
rect
A rectangle.

Return Value
The dictionary representation of the rectangle.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGGeometry.h`

CGRectDivide

Divides a source rectangle into two component rectangles.

```
void CGRectDivide (
   CGRect rect,
   CGRect *slice,
   CGRect *remainder,
   CGFloat amount,
   CGRectEdge edge
);
```

Parameters

rect
The source `CGRect` structure.

slice
On input, a pointer to an uninitialized `CGRect` structure. On return, a `CGRect` structure filled in with the specified edge and values that extends the distance beyond the edge specified by the *amount* parameter.

remainder
On input, a pointer to an uninitialized rectangle `CGRect` structure. On return, the `CGRect` structure contains the portion of the source `CGRect` structure that remains after `CGRectEdge` produces the "slice" rectangle.

amount
A distance from the rectangle side that is specified in the *edge* parameter. This distance defines the line, parallel to the specified side, that Quartz uses to divide the source `CGRect` structure.

edge
A `CGRectEdge` value (`CGRectMinXEdge` (page 2376), `CGRectMinYEdge` (page 2376), `CGRectMaxXEdge` (page 2376), or `CGRectMaxYEdge` (page 2376)) that specifies the side of the rectangle from which the distance passed in the `amount` parameter is measured. `CGRectDivide` produces a "slice" rectangle that contains the specified edge and extends `amount` distance beyond it.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGGeometry.h`

CGRectEqualToRect

Returns whether two rectangles are equal in size and position.

```
bool CGRectEqualToRect (
   CGRect rect1,
   CGRect rect2
);
```

Parameters

rect1

The first rectangle to examine.

rect2

The second rectangle to examine.

Return Value

Returns `1` if the two specified rectangles have equal size and origin values, or are both `null`. Otherwise, returns `0`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGGeometry.h`

CGRectGetHeight

Returns the height of a rectangle.

```
CGFloat CGRectGetHeight (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value

The height of the specified rectangle.

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

CarbonSketch

HID Calibrator

HID Config Save

HID Explorer

WhackedTV

Declared In

`CGGeometry.h`

CGRectGetMaxX

Returns the x-coordinate that establishes the right edge of a rectangle.

```
CGFloat CGRectGetMaxX (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value
The x-coordinate of the top-right corner of the specified rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
HID Calibrator
HID Explorer

Declared In
CGGeometry.h

CGRectGetMaxY

Returns the y-coordinate that establishes the top edge of a rectangle.

```
CGFloat CGRectGetMaxY (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value
The y-coordinate of the top-right corner of the specified rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
HID Explorer

Declared In
CGGeometry.h

CGRectGetMidX

Returns the x- coordinate that establishes the center of a rectangle.

```
CGFloat CGRectGetMidX (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value
The x-coordinate of the center of the specified rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
HID Calibrator

Declared In
`CGGeometry.h`

CGRectGetMidY

Returns the y-coordinate that establishes the center of a rectangle.

```
CGFloat CGRectGetMidY (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value
The y-coordinate of the center of the specified rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
HID Calibrator
HID Explorer

Declared In
`CGGeometry.h`

CGRectGetMinX

Returns the x-coordinate that establishes the left edge of a rectangle.

```
CGFloat CGRectGetMinX (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value
The x-coordinate of the bottom-left corner of the specified rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch
HID Config Save
HID Explorer

Declared In
CGGeometry.h

CGRectGetMinY

Returns the y-coordinate that establishes the bottom edge of a rectangle.

```
CGFloat CGRectGetMinY (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value
The y-coordinate of the bottom-left corner of the specified rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch
HID Config Save
HID Explorer

Declared In
CGGeometry.h

CGRectGetWidth

Returns the width of a rectangle.

```
CGFloat CGRectGetWidth (
   CGRect rect
);
```

Parameters

rect

The rectangle to examine.

Return Value
The width of the specified rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch
HID Calibrator
HID Config Save
HID Explorer
WhackedTV

Declared In
`CGGeometry.h`

CGRectInset

Returns a rectangle that is smaller or larger than the source rectangle, with the same center point.

```
CGRect CGRectInset (
   CGRect rect,
   CGFloat dx,
   CGFloat dy
);
```

Parameters

rect

The source `CGRect` structure.

dx

The x-coordinate value to use for adjusting the source rectangle. To create an inset rectangle, specify a positive value. To create a larger, encompassing rectangle, specify a negative value.

dy

The y-coordinate value to use for adjusting the source rectangle. To create an inset rectangle, specify a positive value. To create a larger, encompassing rectangle, specify a negative value.

Return Value
A filled-in `CGRect` structure. The origin value is offset in the x-axis by the distance specified by the `dx` parameter and in the y-axis by the distance specified by the `dy` parameter, and its size adjusted by `(2*dx,2*dy)`, relative to the source rectangle. If `dx` and `dy` are positive values, then the rectangle's size is decreased. If `dx` and `dy` are negative values, the rectangle's size is increased.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGGeometry.h`

CGRectIntegral

Returns the smallest rectangle that results from converting the source rectangle values to integers.

```
CGRect CGRectIntegral (
   CGRect rect
);
```

Parameters

rect
The source rectangle.

Return Value
A filled-in `CGRect` structure whose values represent the rectangle with the smallest integer values for its origin and size that contains the source rectangle. That is, given a rectangle with fractional origin or size values, `CGRectIntegral` rounds the rectangle's origin downward and its size upward to the nearest whole integers, such that the result contains the original rectangle.

Availability
Available in Mac OS X version 10.0 and later.

See Also
`CGRectIsIntegral` (page 2367)

Related Sample Code
WhackedTV

Declared In
`CGGeometry.h`

CGRectIntersection

Returns the intersection of two rectangles.

```
CGRect CGRectIntersection (
   CGRect r1,
   CGRect r2
);
```

Parameters

rect1
The first source rectangle.

rect2
The second source rectangle.

Return Value
A filled-in `CGRect` structure that represents the intersection of the two specified rectangles. If the two rectangles do not intersect, returns the null rectangle. To check for this condition, use `CGRectIsNull` (page 2368).

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
WhackedTV

Declared In
`CGGeometry.h`

CGRectIntersectsRect

Returns whether two rectangles intersect.

```
bool CGRectIntersectsRect (
   CGRect rect1,
   CGRect rect2
);
```

Parameters

rect1
The first rectangle to examine.

rect2
The second rectangle to examine.

Return Value
Returns `1` if the two specified rectangles intersect; otherwise, `0`. The first rectangle intersects the second if the intersection of the rectangles is not equal to the null rectangle.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGGeometry.h`

CGRectIsEmpty

Returns whether a rectangle has zero width or height, or is a null rectangle.

```
bool CGRectIsEmpty (
   CGRect rect
);
```

Parameters

rect
The rectangle to examine.

Return Value
Returns `1` if the specified rectangle is empty; otherwise, `0`.

Discussion
An empty rectangle is either a null rectangle or a valid rectangle with zero height or width. See also `CGRectIsNull` (page 2368).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGGeometry.h`

CGRectIsInfinite

Returns whether a rectangle is infinite.

```
bool CGRectIsInfinite (
    CGRect rect
);
```

Parameters

rect
The rectangle to examine.

Return Value
Returns `true` if the specified rectangle is infinite, `false` otherwise.

Discussion
An infinite rectangle is one that has no defined bounds. Infinite rectangles can be created as output from a tiling filter. For example, the Core Image framework perspective tile filter creates an image whose extent is described by an infinite rectangle.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
WhackedTV

Declared In
`CGGeometry.h`

CGRectIsIntegral

Returns whether the origin and size of the rectangle can be represented exactly as integers.

```
bool CGRectIsIntegral (
   CGRect rect
);
```

Parameters

rect
The rectangle to examine.

Return Value
Returns `true` if the origin and size of the rectangle can be represented exactly as integers; `false` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGGeometry.h`

CGRectIsNull

Returns whether a rectangle is invalid.

```
bool CGRectIsNull (
   CGRect rect
);
```

Parameters

rect
The rectangle to examine.

Return Value
Returns `1` if the specified rectangle is null; otherwise, `0`.

Discussion
A null rectangle is one that is not valid (you cannot draw a null rectangle). For example, the result of intersecting two disjoint rectangles is a null rectangle. See also `CGRectIsEmpty` (page 2366).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGGeometry.h`

CGRectMake

Returns a `CGRect` structure filled in with the coordinate and dimension values you provide.

```
CGRect CGRectMake (
   CGFloat x,
   CGFloat y,
   CGFloat width,
   CGFloat height
);
```

Parameters

x
The x-coordinate of the rectangle's origin point.

y
The y-coordinate of the rectangle's origin point.

width
The width of the rectangle.

height
The height of the rectangle.

Return Value
Returns a rectangle with the specified location and dimensions.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CALayerEssentials
CarbonSketch
HID Calibrator
HID Explorer
QTCarbonShell

Declared In
`CGGeometry.h`

CGRectMakeWithDictionaryRepresentation

Fills in a `CGRect` structure using the contents of the provided dictionary.

```
bool CGRectMakeWithDictionaryRepresentation(
    CFDictionaryRef dict,
    CGRect *rect
);
```

Parameters

dict
A dictionary that was previously returned from the function `CGRectCreateDictionaryRepresentation` (page 2358).

rect
On return, the rectangle created from the provided dictionary.

Return Value
`true` if successful; `false` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`CGGeometry.h`

CGRectOffset

Returns a rectangle with an origin that is offset from that of the source rectangle.

```
CGRect CGRectOffset (
   CGRect rect,
   CGFloat dx,
   CGFloat dy
);
```

Parameters

rect

The source rectangle.

dx

The offset value for the x-coordinate.

dy

The offset value for the y-coordinate.

Return Value
A filled-in `CGRect` structure that is the same size as the source, but with its origin offset by `dx` units along the x-axis and `dy` units along the y-axis with respect to the source.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGGeometry.h`

CGRectStandardize

Returns a rectangle with a positive width and height.

```
CGRect CGRectStandardize (
   CGRect rect
);
```

Parameters

rect

The source rectangle.

Return Value
A filled-in `CGRect` structure that represents the source rectangle, but with positive width and height values.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGGeometry.h`

CGRectUnion

Returns the smallest rectangle that contains the two provided rectangles.

```
CGRect CGRectUnion (
   CGRect r1,
   CGRect r2
);
```

Parameters

r1

The first source rectangle.

r2

The second source rectangle.

Return Value

A filled-in `CGRect` structure that represents the smallest rectangle that completely contains both of the source rectangles.

Discussion

If one of the rectangles has 0 (or negative) width or height, a copy of the other rectangle is returned; but if both have 0 (or negative) width or height, the returned rectangle has its origin at (0.0, 0.0) and has 0 width and height.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`CGGeometry.h`

CGSizeCreateDictionaryRepresentation

Returns a dictionary representation of the provided size.

```
CFDictionaryRef CGSizeCreateDictionaryRepresentation(
    CGSize size
);
```

Parameters

size

A size.

Return Value

The dictionary representation of the size.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CGGeometry.h`

CGSizeEqualToSize

Returns whether two sizes are equal.

```
bool CGSizeEqualToSize (
   CGSize size1,
   CGSize size2
);
```

Parameters

size1

The first size to examine.

size2

The second size to examine.

Return Value
Returns `1` if the two specified sizes are equal; otherwise, `0`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`CGGeometry.h`

CGSizeMake

Returns a `CGSize` structure filled in with dimension values you provide.

```
CGSize CGSizeMake (
   CGFloat width,
   CGFloat height
);
```

Parameters

width

A width value.

height

A height value.

Return Value
Returns a `CGSize` structure with the specified width and height.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`CGGeometry.h`

CGSizeMakeWithDictionaryRepresentation

Fills in a `CGSize` structure using the contents of the provided dictionary.

```
bool CGSizeMakeWithDictionaryRepresentation(
    CFDictionaryRef dict,
    CGSize *size
);
```

Parameters

dict

A dictionary that was previously returned from the function `CGSizeCreateDictionaryRepresentation` (page 2371).

size

On return, the size created from the provided dictionary.

Return Value

`true` if successful; `false` otherwise.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CGGeometry.h`

Data Types

CGPoint

A structure that contains a point in a two-dimensional coordinate system.

```
struct CGPoint {
     CGFloat x;
    CGFloat y;
};
typedef struct CGPoint CGPoint;
```

Fields

`x`

The x-coordinate of the point.

`y`

The y-coordinate of the point.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGGeometry.h`

CGRect

A structure that contains the location and dimensions of a rectangle.

```
struct CGRect {
    CGPoint origin;
    CGSize size;
};
typedef struct CGRect CGRect;
```

Fields

`origin`
A `CGPoint` (page 2373) structure that specifies the coordinates of the rectangle's origin. The origin is located in the lower-left of the rectangle.

`size`
A `CGSize` (page 2374) structure that specifies the height and width of the rectangle.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGGeometry.h`

CGSize

A structure that contains width and height values.

```
struct CGSize {
    CGFloat width;
    CGFloat height;
};
typedef struct CGSize CGSize;
```

Fields

`width`
A width value.

`height`
A height value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGGeometry.h`

Constants

CGRectInfinite

A rectangle that has infinite extent.

```
const CGRect CGRectInfinite;
```

Constants

`CGRectInfinite`

A rectangle that has infinite extent.

Available in Mac OS X v10.4 and later.

Declared in `CGGeometry.h`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGGeometry.h`

Geometric Zeroes

A zero point, zero rectangle, or zero size.

```
const CGPoint CGPointZero;
const CGRect CGRectZero;
const CGSize CGSizeZero;
```

Constants

`CGPointZero`

A point constant with location `(0, 0)`. The zero point is equivalent to `CGPointMake(0,0)`.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

`CGRectZero`

A rectangle constant with location `(0,0)`, and width and height of 0. The zero rectangle is equivalent to `CGRectMake(0,0,0,0)`.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

`CGSizeZero`

A size constant with width and height of *0*. The zero size is equivalent to `CGSizeMake(0,0)`.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

Declared In

`CGGeometry.h`

Geometrical Null

The null or empty rectangle.

```
const CGRect CGRectNull;
```

Constants

`CGRectNull`

The null rectangle. This is the rectangle returned when, for example, you intersect two disjoint rectangles. Note that the null rectangle is not the same as the zero rectangle.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

Declared In

`CGGeometry.h`

CGRectEdge

Coordinates that establish the edges of a rectangle.

```
enum CGRectEdge {
    CGRectMinXEdge,
    CGRectMinYEdge,
    CGRectMaxXEdge,
    CGRectMaxYEdge
};
typedef enum CGRectEdge CGRectEdge;
```

Constants

`CGRectMinXEdge`

The x-coordinate that establishes the left edge of a rectangle.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

`CGRectMinYEdge`

The y-coordinate that establishes the minimum edge of a rectangle. In Mac OS X, this is typically the bottom edge of the rectangle. If the coordinate system is flipped (or if you are using the default coordinate system in iPhone OS), this constant refers to the top edge of the rectangle.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

`CGRectMaxXEdge`

The x-coordinate that establishes the right edge of a rectangle.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

`CGRectMaxYEdge`

The y-coordinate that establishes the maximum edge of a rectangle. In Mac OS X, this is typically the top edge of the rectangle. If the coordinate system is flipped (or if you are using the default coordinate system in iPhone OS), this constant refers to the bottom edge of the rectangle.

Available in Mac OS X v10.0 and later.

Declared in `CGGeometry.h`.

Declared In

`CGGeometry.h`

CGFloat Informational Macros

Informational macros for the `CGFloat` type.

```
#define CGFLOAT_MIN FLT_MIN                                     // 32-bit
#define CGFLOAT_MAX FLT_MAX
#define CGFLOAT_IS_DOUBLE 0

#define CGFLOAT_MIN DBL_MIN                                     // 64-bit
#define CGFLOAT_MAX DBL_MAX
#define CGFLOAT_IS_DOUBLE 1
```

Constants

`CGFLOAT_MIN`

The minimum allowable value for a `CGFloat` type. For 32-bit code, this value is `1.17549435e-38F`. For 64-bit code, it is `2.2250738585072014e-308`.

Available in Mac OS X v10.5 and later.

Declared in `CABase.h`.

`CGFLOAT_MAX`

The maximum allowable value for a `CGFloat` type. For 32-bit code, this value is `3.40282347e+38F`. For 64-bit code, it is `1.7976931348623157e+308`.

Available in Mac OS X v10.5 and later.

Declared in `CABase.h`.

`CGFLOAT_IS_DOUBLE`

Indicates whether `CGFloat` is defined as a `float` or `double` type.

Available in Mac OS X v10.5 and later.

Declared in `CABase.h`.

Find By Content Reference (Not Recommended)

Framework: ApplicationServices/ApplicationServices.h

Declared in FindByContent.h

Overview

Important: The Find by Content API is deprecated as of Mac OS X v10.4. A much more complete solution for finding and displaying information is provided by the Search Kit. See *Search Kit Programming Guide* for guidelines on using the Search Kit.

Whereas the Find by Content API searches specified volumes or folders for words typed in by a user, Search Kit uses a much faster fully indexed search to return relevant content of all sorts. Because the Search Kit takes a different approach to searching for and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Seach Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Previous to Mac OS X version 10.2, the main client for Find by Content was Sherlock; in versions 10.2 and 10.3 it was the Finder. In version 10.4 and later, Search Kit is used instead by the Finder, Mail, and Spotlight.

Carbon supports Find By Content, but note that it is not contained within the Carbon framework.

Functions by Task

Working With Indexes

`FBCDeleteIndexFileForFolder` (page 2389) Deprecated in Mac OS X v10.4

Deletes the index file associated with a folder. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCIndexItemsInLanguages` (page 2403) Deprecated in Mac OS X v10.4

Indexes one or more directories. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Setting Up a Search Session Object

`FBCCreateSearchSession` (page 2388) Deprecated in Mac OS X v10.4
Creates a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCSetSessionCallback` (page 2406) Deprecated in Mac OS X v10.4
Sets a callback function that allows your application to cancel a search operation. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCSetSessionHitTest` (page 2407) Deprecated in Mac OS X v10.4
Sets a callback function that allows your application to perform search-hit testing. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Executing a Search

`FBCBlindExampleSearchWithCallback` (page 2386) Deprecated in Mac OS X v10.4
Performs a similarity search that uses example files instead of a query string, and installs callbacks for progress reporting and search hit testing. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCDoCFStringSearch` (page 2391) Deprecated in Mac OS X v10.4
Initiates a search using a `CFString` as the query string. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCDoExampleSearch` (page 2393) Deprecated in Mac OS X v10.4
Performs a similarity search that uses example files instead of a query string. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCDoQuerySearch` (page 2394) Deprecated in Mac OS X v10.4
Initiates a search using a C string as the query string. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Getting Information About Search Session Hits

`FBCGetHitCount` (page 2395) Deprecated in Mac OS X v10.4
Obtains the number of hits returned for a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCGetHitDocument` (page 2396) Deprecated in Mac OS X v10.4
Retrieves the `FSSpec` associated with a specific hit and search. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCGetHitDocumentRef` (page 2397) Deprecated in Mac OS X v10.4
Retrieves the `FSRef` associated with a specific hit and search. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCGetHitScore` (page 2397) Deprecated in Mac OS X v10.4
Obtains the hit score associated with a specific hit. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Deallocating Hit Lists and Search Sessions

`FBCDestroySearchSession` (page 2390) Deprecated in Mac OS X v10.4
Disposes of a search session object. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCReleaseSessionHits` (page 2404) Deprecated in Mac OS X v10.4
Releases the hits associated with a search session object. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Summarizing Text

`FBCDisposeSummary` (page 2391) Deprecated in Mac OS X v10.4
Disposes of a summary reference object. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCGetSummaryOfCFString` (page 2400) Deprecated in Mac OS X v10.4
Creates a summary reference object for `CFString`. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCGetSummarySentenceCount` (page 2400) Deprecated in Mac OS X v10.4
Obtains the number of sentences in a summary reference object. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCGetSummarySentences` (page 2401) Deprecated in Mac OS X v10.4
Obtains a summary that contains a specified number of sentences. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCSummarize` (page 2408) Deprecated in Mac OS X v10.4
Summarizes text that is specified as an ASCII buffer. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCSummarizeCFString` (page 2409) Deprecated in Mac OS X v10.4
Summarizes text that is specified as a `CFString`. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Working with Universal Procedure Pointers

`DisposeFBCCallbackUPP` (page 2383) Deprecated in Mac OS X v10.4
Disposes of universal procedure pointer (UPP) to a search cancellation callback. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`DisposeFBCHitTestUPP` (page 2384) Deprecated in Mac OS X v10.4
Disposes of universal procedure pointer (UPP) to a search-hit testing callback. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`InvokeFBCCallbackUPP` (page 2412) Deprecated in Mac OS X v10.4
Invokes a universal procedure pointer (UPP) to a search cancellation callback. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`InvokeFBCHitTestUPP` (page 2413) Deprecated in Mac OS X v10.4
Invokes a universal procedure pointer (UPP) to a search-hit testing callback. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

NewFBCCallbackUPP (page 2413) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a search cancellation callback. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

NewFBCHitTestUPP (page 2414) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a search-hit testing callback. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Deprecated Functions

FBCAddAllVolumesToSession (page 2384) Deprecated in Mac OS X v10.4
Adds all volumes to a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCAddVolumeToSession (page 2385) Deprecated in Mac OS X v10.4
Adds one volume to a session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCBlindExampleSearch (page 2385) Deprecated in Mac OS X v10.4
Executes a similarity search. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCCloneSearchSession (page 2388) Deprecated in Mac OS X v10.4
Clones a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCDestroyWordList (page 2390) Deprecated in Mac OS X v10.4
Destroys a word list. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCFindIndexFileFolderForFolder (page 2395) Deprecated in Mac OS X v10.4
Gets the location of the index file associated with a directory. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCGetMatchedWords (page 2398) Deprecated in Mac OS X v10.4
Gets a list of matched words for a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCGetSessionVolumeCount (page 2399) Deprecated in Mac OS X v10.4
Gets a volume count for a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCGetSessionVolumes (page 2399) Deprecated in Mac OS X v10.4
Gets the volumes associated with a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCGetTopicWords (page 2402) Deprecated in Mac OS X v10.4
Gets a list of topic words for a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCIndexItems (page 2403) Deprecated in Mac OS X v10.4
Indexes one or more files or folders. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCRemoveVolumeFromSession (page 2405) Deprecated in Mac OS X v10.4
Removes a volume from a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCSetCallback (page 2405) Deprecated in Mac OS X v10.4
Sets a callback. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

FBCSetHeapReservation (page 2406) Deprecated in Mac OS X v10.4
Sets the amount of heap space to reserve for application use when Find by Content allocates memory. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCSetSessionVolumes` (page 2408) Deprecated in Mac OS X v10.4
: Sets the volumes for a search session. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCVolumeIndexPhysicalSize` (page 2410) Deprecated in Mac OS X v10.4
: Obtains the physical size of an index. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCVolumeIndexTimeStamp` (page 2410) Deprecated in Mac OS X v10.4
: Obtains the date and time when an index was last updated. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCVolumeIsIndexed` (page 2411) Deprecated in Mac OS X v10.4
: Determines whether a volume is indexed. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

`FBCVolumeIsRemote` (page 2411) Deprecated in Mac OS X v10.4
: Determines whether a volume is remote. (Deprecated. Use Search Kit instead; see *Search Kit Programming Guide*.)

Functions

DisposeFBCCallbackUPP

Disposes of universal procedure pointer (UPP) to a search cancellation callback. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
void DisposeFBCCallbackUPP (
    FBCCallbackUPP userUPP
);
```

Parameters

userUPP
: The universal procedure pointer.

Discussion

See the callback `FBCCallbackProcPtr` (page 2414) for more information.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

DisposeFBCHitTestUPP

Disposes of universal procedure pointer (UPP) to a search-hit testing callback. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
void DisposeFBCHitTestUPP (
    FBCHitTestUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `FBCHitTestProcPtr` (page 2415) for more information.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.2 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCAddAllVolumesToSession

Adds all volumes to a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCAddAllVolumesToSession (
    FBCSearchSession theSession,
    Boolean includeRemote
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCAddVolumeToSession

Adds one volume to a session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCAddVolumeToSession (
    FBCSearchSession theSession,
    SInt16 vRefNum
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCBlindExampleSearch

Executes a similarity search. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCBlindExampleSearch (
    const FSSpec examples[],
    UInt32 numExamples,
    const FSSpec targetDirs[],
    UInt32 numTargets,
    UInt32 maxHits,
    UInt32 maxHitWords,
    Boolean allIndexes,
    Boolean includeRemote,
    FBCSearchSession * theSession
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

Use the function `FBCBlindExampleSearchWithCallback` (page 2386) instead.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCBlindExampleSearchWithCallback

Performs a similarity search that uses example files instead of a query string, and installs callbacks for progress reporting and search hit testing. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCBlindExampleSearchWithCallback (
    const FSSpec examples[],
    UInt32 numExamples,
    const FSSpec targetDirs[],
    UInt32 numTargets,
    UInt32 maxHits,
    UInt32 maxHitWords,
    Boolean allIndexes,
    Boolean includeRemote,
    FBCSearchSession * theSession,
    FBCCallbackUPP callback,
    void * callbackData,
    FBCHitTestUPP userHitTest,
    void * userHitTestData
);
```

Parameters

examples

An array of `FSSpec` data types that specify the location of one or more examples files. The hits are the files that most resemble the examples in terms of weighted word frequencies.

numExamples
The number of examples in the `examples` array.

targetDirs
An array of `FSSpec` data types that specify the directories you want to search. Any directories that are not indexed are skipped.

numTargets
A value that specifies number of directories specified by the `targetDirs` parameter.

maxHits
A value that specifies the maximum number of hits you want. If you pass N for this parameter, you get the N most relevant hits found in an exhaustive search of the indexes (or all the hits if there are less than N).

maxHitWords
This parameter is ignored in Mac OS X version 10.2 and later.

allIndexes
This parameter is ignored in Mac OS X version 10.2 and later.

includeRemote
This parameter is ignored in Mac OS X version 10.2 and later.

theSession
A valid search session object. You obtain a search session object by calling the function `FBCCreateSearchSession`.

callback
A universal procedure pointer to your callback for cancelling a search. See `FBCCallbackProcPtr` (page 2414) for more information.

callbackData
A pointer to data needed by the callback specified by the `callback` parameter. Pass `NULL` if your callback does not require any data.

userHitTest
A universal procedure pointer a your callback for testing search hits. See `FBCHitTestProcPtr` (page 2415) for more information.

callbackData
A pointer to data needed by the callback specified by the `userHitTest` parameter. Pass `NULL` if your callback does not require any data.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Discussion
You should use the function `FBCBlindExampleSearchWithCallback` for cases where your example files have not been indexed. If the files have been indexed, it is simpler and more efficient for you to use the function `FBCDoExampleSearch`.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCCloneSearchSession

Clones a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCCloneSearchSession (
    FBCSearchSession original,
    FBCSearchSession * clone
);
```

Parameters

original

A valid search session object. You obtain a search session object by calling the function `FBCCreateSearchSession`.

clone

On return, points to a newly-created search session object that is a copy of the one specified by the `original` parameter.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCCreateSearchSession

Creates a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCCreateSearchSession (
    FBCSearchSession * searchSession
);
```

Parameters

searchSession

On return, points to a newly-created search session object. You provide a search session object as a parameter when you call other other search functions.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Discussion
You must create a search session using the function `FBCCreateSearchSession` before you can use any other search function.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCDeleteIndexFileForFolder

Deletes the index file associated with a folder. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCDeleteIndexFileForFolder (
    const FSRef * folder
);
```

Parameters

folder

An `FSRef` that specifies the folder whose index file you want to delete.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCDestroySearchSession

Disposes of a search session object. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCDestroySearchSession (
    FBCSearchSession theSession
);
```

Parameters

`theSession`

The search session object you want to dispose of. You obtain a search session object by calling the function `FBCCreateSearchSession`.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCDestroyWordList

Destroys a word list. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCDestroyWordList (
    FBCWordList theList,
    UInt32 wordCount
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCDisposeSummary

Disposes of a summary reference object. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSStatus FBCDisposeSummary (
    FBCSummaryRef summary
);
```

Parameters

summary

The summary reference object you want to dispose of. You obtain a summary reference object when you call the function `FBCGetSummaryOfCFString` (page 2400).

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Discussion
You must dispose of a summary reference object when you no longer need it.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCDoCFStringSearch

Initiates a search using a `CFString` as the query string. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCDoCFStringSearch (
    FBCSearchSession theSession,
    CFStringRef queryString,
    const FSSpec targetDirs[],
    UInt32 numTargets,
    UInt32 maxHits,
    UInt32 maxHitWords
);
```

Parameters

theSession

A valid search session object. You obtain a search session object by calling the function `FBCCreateSearchSession`.

queryString

The query string specified as a `CFString`.

targetDirs

An array of `FSSpec` data types that specify the directories you want to search. Any directories that are not indexed are skipped.

numTargets

A value that specifies number of directories specified by the `targetDirs` parameter.

maxHits

A value that specifies the maximum number of hits you want. If you pass N for this parameter, you get the N most relevant hits found in an exhaustive search of the indexes (or all the hits if there are less than N).

maxHitWords

This parameter is ignored in Mac OS X version 10.2 and later.

Return Value

A result code. See "Find By Content Result Codes" (page 2421).

Discussion

When the function `FBCDoCFStringSearch` returns, you must retrieve hit information by calling the functions `FBCGetHitCount` (page 2395), `FBCGetHitDocumentRef` (page 2397), and `FBCGetHitScore` (page 2397). You call `FBCGetHitCount` (page 2395) to obtain the number of hits found. Then you can iterate through each hit to get the associated `FSRef` and score by calling `FBCGetHitDocumentRef` (page 2397) and `FBCGetHitScore` (page 2397).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCDoExampleSearch

Performs a similarity search that uses example files instead of a query string. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCDoExampleSearch (
    FBCSearchSession theSession,
    const UInt32 * exampleHitNums,
    UInt32 numExamples,
    const FSSpec targetDirs[],
    UInt32 numTargets,
    UInt32 maxHits,
    UInt32 maxHitWords
);
```

Parameters

theSession

A valid search session object. You must provide the search session object you provided to the search function `FBCDoCFStringSearch`.

exampleHitNums

A pointer to the hits obtained from a previous search using `theSession` search session object.

numExamples

The number of examples pointed to by the `exampleHitNums` parameter.

targetDirs

An array of `FSSpec` data types that specify the directories you want to search. Any directories that are not indexed are skipped.

numTargets

A value that specifies number of directories specified by the `targetDirs` parameter.

maxHits

A value that specifies the maximum number of hits you want. If you pass N for this parameter, you get the N most relevant hits found in an exhaustive search of the indexes (or all the hits if there are less than N).

maxHitWords

This parameter is ignored in Mac OS X version 10.2 and later.

Return Value

A result code. See "Find By Content Result Codes" (page 2421).

Discussion

You should use the function `FBCDoExampleSearch` for cases where you example files have been indexed. If you example files have not been indexed, us the function `FBCBlindExampleSearchWithCallback`.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCDoQuerySearch

Initiates a search using a C string as the query string. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCDoQuerySearch (
    FBCSearchSession theSession,
    char * queryText,
    const FSSpec targetDirs[],
    UInt32 numTargets,
    UInt32 maxHits,
    UInt32 maxHitWords
);
```

Parameters

theSession
A valid search session object. You obtain a search session object by calling the function `FBCCreateSearchSession`.

queryText
The query string specified as a C string.

numTargets
A value that specifies number of directories specified by the `targetDirs` parameter.

maxHits
A value that specifies the maximum number of hits you want. If you pass N for this parameter, you get the N most relevant hits found in an exhaustive search of the indexes (or all the hits if there are less than N).

maxHitWords
This parameter is ignored in Mac OS X version 10.2 and later.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Discussion
You should use the function `FBCDoCFStringSearch` (page 2391) instead of this one.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCFindIndexFileFolderForFolder

Gets the location of the index file associated with a directory. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCFindIndexFileFolderForFolder (
    const FSRef * inFolder,
    FSRef * outFolder
);
```

Parameters

inFolder

A pointer to an `FSRef` that specifies the location of the folder that contains the index file for a directory.

outFolder

On output, points to the `FSRef` that specifies the directory that contains he index file.

Return Value

A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCGetHitCount

Obtains the number of hits returned for a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCGetHitCount (
    FBCSearchSession theSession,
    UInt32 * count
);
```

Parameters

theSession

A valid search session object. You must provide the search session object you provided to the search function `FBCDoCFStringSearch`.

count

On return, points to the number of hits returned for the search on the object specified by `theSession`.

Return Value

A result code. See "Find By Content Result Codes" (page 2421).

Discussion
You can iterate through each hit to get the associated `FSRef` and score by calling the functions `FBCGetHitDocumentRef` (page 2397) and `FBCGetHitScore` (page 2397).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCGetHitDocument

Retrieves the `FSSpec` associated with a specific hit and search. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCGetHitDocument (
    FBCSearchSession theSession,
    UInt32 hitNumber,
    FSSpec * theDocument
);
```

Parameters

`theSession`
A valid search session object. You must provide the search session object you provided to the search function `FBCDoCFStringSearch`.

`hitNumber`
A value that specifies the hit whose document you want to retrieve.

`theDocument`
A pointer to an `FSSpec` that specifies the location of the document associated with `hitNumber` for the search previously carried out on `theSession`.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCGetHitDocumentRef

Retrieves the `FSRef` associated with a specific hit and search. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCGetHitDocumentRef (
    FBCSearchSession theSession,
    UInt32 hitNumber,
    FSRef * theDocument
);
```

Parameters

theSession

A valid search session object. You must provide the search session object you provided to the search function `FBCDoCFStringSearch`.

hitNumber

A value that specifies the hit whose document you want to retrieve.

theDocument

A pointer to an `FSRef` that specifies the location of the document associated with `hitNumber` for the search previously carried out on `theSession`.

Return Value

A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCGetHitScore

Obtains the hit score associated with a specific hit. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCGetHitScore (
    FBCSearchSession theSession,
    UInt32 hitNumber,
    float * score
);
```

Parameters

theSession

A valid search session object. You must provide the search session object you provided to the search function `FBCDoCFStringSearch`.

hitNumber

A value that specifies the hit whose document you want to retrieve.

float

A pointer to a value that specifies the hit score. Scores are normalized to range from 1.0 (most significant) to 0 (least significant).

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCGetMatchedWords

Gets a list of matched words for a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCGetMatchedWords (
    FBCSearchSession theSession,
    UInt32 hitNumber,
    UInt32 * wordCount,
    FBCWordList * list
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCGetSessionVolumeCount

Gets a volume count for a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCGetSessionVolumeCount (
    FBCSearchSession theSession,
    UInt16 * count
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCGetSessionVolumes

Gets the volumes associated with a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCGetSessionVolumes (
    FBCSearchSession theSession,
    SInt16 vRefNums[],
    UInt16 * numVolumes
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCGetSummaryOfCFString

Creates a summary reference object for `CFString`. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSStatus FBCGetSummaryOfCFString (
    CFStringRef inString,
    FBCSummaryRef * summary
);
```

Parameters

`inString`

A `CFStringRef` representing the text you want summarized. Summarization works with all languages that use white space to separate words, plus Japanese. It works best with text that is not broken into lines using CR and/or LF characters.

`summary`

On output, a newly-created summary reference object that contains summary information for the string specified by the `inString` parameter. A summary reference object is an opaque object. To access the information in this object, you can use the functions `FBCGetSummarySentenceCount` (page 2400) and `FBCGetSummarySentences` (page 2401). You should call the function `FBCDisposeSummary` (page 2391) when you no longer need the summary reference object.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Discussion
A summary reference object is an opaque object that can very quickly give you a summary containing any desired number of sentences from 1 up to the total number of sentences found in the original text.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCGetSummarySentenceCount

Obtains the number of sentences in a summary reference object. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSStatus FBCGetSummarySentenceCount (
    FBCSummaryRef summary,
    UInt32 * numSentences
);
```

Parameters

summary

A valid summary reference object. You obtain a summary reference object when you call the function `FBCGetSummaryOfCFString` (page 2400).

numSentences

On output, a pointer to the number of sentences contained in the summary.

Return Value

A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.2 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCGetSummarySentences

Obtains a summary that contains a specified number of sentences. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSStatus FBCGetSummarySentences (
    FBCSummaryRef summary,
    CFStringRef * outString,
    UInt32 * numSentences,
    Boolean paragraphs
);
```

Parameters

summary

A valid summary reference object. You obtain a summary reference object when you call the function `FBCGetSummaryOfCFString` (page 2400).

outString

On output, a pointer to a `CFStringRef` that contains the summary. You are responsible for releasing the `CFStringRef` when you no longer need it.

numSentences

On input, the number of sentences you want in the summary. On output, the number of sentences actually written into the summary. If you pass `0`, Find by Content uses a logarithmic function to determine the number of sentences.

paragraphs

A Boolean value that specifies the content of the summary. If you pass `true` the summary is made up of whole paragraphs that contain the relevant sentences. If it is `false`, only the relevant sentences are included.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCGetTopicWords

Gets a list of topic words for a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCGetTopicWords (
    FBCSearchSession theSession,
    UInt32 hitNumber,
    UInt32 * wordCount,
    FBCWordList * list
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCIndexItems

Indexes one or more files or folders. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCIndexItems (
    FSSpecArrayPtr theItems,
    UInt32 itemCount
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCIndexItemsInLanguages

Indexes one or more directories. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCIndexItemsInLanguages (
    FSSpecArrayPtr theItems,
    UInt32 itemCount,
    UInt32 languageHighBits,
    UInt32 languageLowBits
);
```

Parameters

`theItems`

An `FSPecArrayPtr` that specifies the directory or directories to be indexed; subdirectories are included automatically.

`itemCount`

The number of items pointed to by `theItems` parameter.

`languageHighBits`

A value that specifies the languages to be used. The high bits are obtained by summing the desired constants defined in Language Constants (page 2418).

`languageLowBits`

A value that specifies the languages to be used. The low bits should always be 0.

Return Value

A result code. See "Find By Content Result Codes" (page 2421).

Discussion
Your application must first call the function `FBCIndexItemsInLanguages` to make an index of the files to be searched. Find by Content looks at the contents of each file (including the file's name) to make the index. For PDF and HTML files, the textual content is separated out from formatting information; for all other file types, the entire data of the file is used. The index is an invisible file.

Some file types do not contain words that are useful for searching, such as graphics files and application software. Find by Content has lists of Mac OS file types and filename extensions for such files, and any file that has one of the listed types or one of the listed extensions is treated specially when it is indexed. That is, the name of the file is indexed, but the contents are not. In addition, folder names are indexed. The lists of types and extensions are text files named StopTypes and StopExts. You can edit these lists using an editor (such as BBEdit or TextEdit). The files are located in the `/System/Library/Find/` directory.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCReleaseSessionHits

Releases the hits associated with a search session object. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCReleaseSessionHits (
    FBCSearchSession theSession
);
```

Parameters

theSession
A valid search session object. You obtain a search session object by calling the function `FBCCreateSearchSession`.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Discussion
The function `FBCReleaseSessionHits` releases the hits associated with a specific search session object, resetting the object so it can be used in another search. You are responsible for releasing a search session object when you no longer need it by calling the function `FBCDestroySearchSession` (page 2390).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCRemoveVolumeFromSession

Removes a volume from a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCRemoveVolumeFromSession (
    FBCSearchSession theSession,
    SInt16 vRefNum
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCSetCallback

Sets a callback. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not Recommended.

```
void FBCSetCallback (
    FBCCallbackUPP fn,
    void * data
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

Use the function `FBCSetSessionCallback` (page 2406) instead.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCSetHeapReservation

Sets the amount of heap space to reserve for application use when Find by Content allocates memory. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
void FBCSetHeapReservation (
    UInt32 bytes
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCSetSessionCallback

Sets a callback function that allows your application to cancel a search operation. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
void FBCSetSessionCallback (
    FBCSearchSession searchSession,
    FBCCallbackUPP fn,
    void * data
);
```

Parameters

searchSession

A valid search session object. You obtain a search session object by calling the function `FBCCreateSearchSession`.

fn

A universal procedure pointer to your callback for cancelling a search. See `FBCCallbackProcPtr` (page 2414) for more information.

data

A pointer to data needed by your callback. Pass `NULL` if your callback does not require any data.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.2 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCSetSessionHitTest

Sets a callback function that allows your application to perform search-hit testing. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
void FBCSetSessionHitTest (
    FBCSearchSession theSession,
    FBCHitTestUPP fn,
    void * data
);
```

Parameters

theSession

A valid search session object. You obtain a search session object by calling the function `FBCCreateSearchSession`.

fn

A universal procedure pointer a your callback for testing search-hits. See `FBCHitTestProcPtr` (page 2415) for more information.

data

A pointer to data needed by your callback. Pass `NULL` if your callback does not require any data.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.2 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCSetSessionVolumes

Sets the volumes for a search session. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCSetSessionVolumes (
    FBCSearchSession theSession,
    const SInt16 vRefNums[],
    UInt16 numVolumes
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCSummarize

Summarizes text that is specified as an ASCII buffer. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSErr FBCSummarize (
    const void * inBuf,
    UInt32 inLength,
    void * outBuf,
    UInt32 * outLength,
    UInt32 * numSentences
);
```

Parameters

inBuf

A pointer to the ASCII text you want summarized.

inLength

The length of the text buffer pointed to by `inBuf`.

outBuf

On output, points to the summarized text.

outLength

On output, points to a value that specifies the length of the summarized text.

numSentences

On input, points to a value that specifies the number of sentences desired in the summary. On output, points to the number of sentences actually written into the summary. If you pass `0`, Find by Content uses a logarithmic function to determine the number of sentences.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Discussion
You should use the function `FBCSummarizeCFString` instead of `FBCSummarize` because the `CFString` version is more reliable and accurate.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCSummarizeCFString

Summarizes text that is specified as a `CFString`. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
OSStatus FBCSummarizeCFString (
    CFStringRef inString,
    CFStringRef * outString,
    UInt32 * numSentences
);
```

Parameters

inString

A `CFStringRef` representing the text you want summarized. Summarization works with all languages that use white space to separate words, plus Japanese. It works best with text that is not broken into lines using CR and/or LF characters.

outString

On output, points to the summarization. You are responsible for releasing this string.

numSentences

On input, points to a value that specifies the number of sentences you want in the summary. On output, points to the number of sentences actually written into the summary. If you pass `0`, Find by Content uses a logarithmic function to determine the number of sentences.

Return Value
A result code. See "Find By Content Result Codes" (page 2421).

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.2 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCVolumeIndexPhysicalSize

Obtains the physical size of an index. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCVolumeIndexPhysicalSize (
    SInt16 theVRefNum,
    UInt32 * size
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCVolumeIndexTimeStamp

Obtains the date and time when an index was last updated. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
OSErr FBCVolumeIndexTimeStamp (
    SInt16 theVRefNum,
    UInt32 * timeStamp
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCVolumeIsIndexed

Determines whether a volume is indexed. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
Boolean FBCVolumeIsIndexed (
    SInt16 theVRefNum
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCVolumeIsRemote

Determines whether a volume is remote. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

Not recommended

```
Boolean FBCVolumeIsRemote (
    SInt16 theVRefNum
);
```

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Carbon Porting Notes

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

InvokeFBCCallbackUPP

Invokes a universal procedure pointer (UPP) to a search cancellation callback. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
Boolean InvokeFBCCallbackUPP (
    UInt16 phase,
    float percentDone,
    void * data,
    FBCCallbackUPP userUPP
);
```

Discussion

You should not need to call this function as Find by Content invokes your callback for you.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

InvokeFBCHitTestUPP

Invokes a universal procedure pointer (UPP) to a search-hit testing callback. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
Boolean InvokeFBCHitTestUPP (
    const FSRef * theFile,
    void * data,
    FBCHitTestUPP userUPP
);
```

Discussion

You should not need to call this function as Find by Content invokes your callback for you.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.2 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In

`FindByContent.h`

NewFBCCallbackUPP

Creates a new universal procedure pointer (UPP) to a search cancellation callback. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
FBCCallbackUPP NewFBCCallbackUPP (
    FBCCallbackProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your progress callback.

Return Value

See the description of the `FBCCallbackUPP` data type.

Discussion

See the callback `FBCCallbackProcPtr` (page 2414) for more information.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

NewFBCHitTestUPP

Creates a new universal procedure pointer (UPP) to a search-hit testing callback. (Deprecated in Mac OS X v10.4. Use Search Kit instead; see *Search Kit Programming Guide*.)

```
FBCHitTestUPP NewFBCHitTestUPP (
    FBCHitTestProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your search-hit-testing callback.

Return Value
See the description of the `FBCHitTestUPP` data type.

Discussion
See the callback `FBCHitTestProcPtr` (page 2415) for more information.

Special Considerations

Because the Search Kit takes a different approach to finding and displaying information from that used by the Find by Content API, you cannot make a one-to-one substitution of Search Kit functions for Find by Content functions. However, the basic features of the Search Kit can be implemented very quickly, and Search Kit offers much greater capability than the Find by Content API.

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.
Deprecated in Mac OS X v10.4.

Declared In
`FindByContent.h`

Callbacks

FBCCallbackProcPtr

Defines a pointer to a function that can cancel a search operation.

```
typedef Boolean (*FBCCallbackProcPtr)
(
    UInt16 phase,
    float percentDone,
    void * data
);
```

If you name your function `MyFBCCallbackProc`, you would declare it like this:

```
Boolean MyFBCCallbackProcPtr (
    UInt16 phase,
```

```
    float percentDone,
    void * data
);
```

Parameters

phase

percentDone

data

A pointer to data needed by your callback. This is the same data you provided to the function `FBCSetSessionCallback` (page 2406) in the `data` parameter.

Return Value

Return `true` if you wants to cancel the current operation. Otherwise return `false`.

Discussion

Find by Content invoke your callback periodically (approximately every 5 ticks) while searching. Your callback can cancel a search operation by returning a value of `true`.

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

`FindByContent.h`

FBCHitTestProcPtr

Defines a pointer to a function that performs search-hit testing.

```
typedef Boolean (*FBCHitTestProcPtr)
(
    const FSRef * theFile,
    void * data
);
```

If you name your function `MyFBCHitTestProc`, you would declare it like this:

```
Boolean MyFBCHitTestProcPtr (
    const FSRef * theFile,
    void * data
);
```

Parameters

theFile

A pointer to an `FSRef` that specifies the location of file that matches the search query.

data

A pointer to data needed by your callback. This is the same data you provided to the function `FBCSetSessionHitTest` (page 2407) in the `data` parameter.

Return Value

Return `true` if your callback wants to accept the file as a valid match or `false` to reject it.

Discussion
You can use this callback to impose additional matching criteria in addition to that provided to Find by Content. For example, you could accept a hit only if the file has a creation date later than a specified date.

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.

Declared In
`FindByContent.h`

Data Types

FBCCallbackUPP

Defines a universal procedure pointer to a search cancellation callback.

```
typedef FBCCallbackProcPtr FBCCallbackUPP;
```

Discussion
For more information, see the description of the `FBCCallbackProcPtr` (page 2414) callback function.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCHitTestUPP

Defines a universal procedure pointer to a search-hit testing callback.

```
typedef FBCHitTestProcPtr FBCHitTestUPP;
```

Discussion
For more information, see the description of the `FBCHitTestProcPtr` (page 2415) callback function.

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCSearchSession

Defines pointer to an opaque data type (referred to as a search session object) that contains a collection of state information used in a search session.

```
typedef struct OpaqueFBCSearchSession * FBCSearchSession;
```

Discussion
You call the function `FBCCreateSearchSession` (page 2388)to create a new search session object. When no longer need the search session object, you must dispose of it by calling the function `FBCDestroySearchSession` (page 2390).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCSummaryRef

A pointer to an opaque data type (referred to as a summary reference object) that contains summary information, from which summary text can be obtained.

```
typedef struct OpaqueFBCSummaryRef * FBCSummaryRef;
```

Discussion
You call the function `FBCGetSummaryOfCFString` (page 2400)to create a new summary reference object. When no longer need the summary reference object, you must dispose of it by calling the function `FBCDisposeSummary` (page 2391).

Availability
Available in Mac OS X v10.2 through Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCWordItem

Defines a data type for an ordinary C string used for searching.

```
typedef char* FBCWordItem;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`FindByContent.h`

FBCWordList

Defines an array of word items.

```
typedef FBCWordItem * FBCWordList;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`FindByContent.h`

Constants

Language Constants

Define values for language constants.

```
enum {
    kFBCenglishHighWord = 0x80000000,
    kFBCdutchHighWord = 0x40000000,
    kFBCgermanHighWord = 0x20000000,
    kFBCswedishHighWord = 0x10000000,
    kFBCdanishHighWord = 0x08000000,
    kFBCspanishHighWord = 0x04000000,
    kFBCportugueseHighWord = 0x02000000,
    kFBCitalianHighWord = 0x01000000,
    kFBCfrenchHighWord = 0x00800000,
    kFBCromanHighWord = 0x00400000,
    kFBCicelandicHighWord = 0x00200000,
    kFBChebrewHighWord = 0x00100000,
    kFBCarabicHighWord = 0x00080000,
    kFBCcenteuroHighWord = 0x00040000,
    kFBCcroatianHighWord = 0x00020000,
    kFBCturkishHighWord = 0x00010000,
    kFBCromanianHighWord = 0x00008000,
    kFBCgreekHighWord = 0x00004000,
    kFBCcyrillicHighWord = 0x00002000,
    kFBCdevanagariHighWord = 0x00001000,
    kFBCgujuratiHighWord = 0x00000800,
    kFBCgurmukhiHighWord = 0x00000400,
    kFBCjapaneseHighWord = 0x00000200,
    kFBCkoreanHighWord = 0x00000100,
    kFBCdefaultLanguagesHighWord = 0xFF800000
};
```

Discussion
Language constants are passed as parameters to the function `FBCIndexItemsInLanguages` (page 2403). The purpose of these constants is to tell Find by Content what languages the user expects the files to contain. From this, Find by Content infer the character encodings to look for, and for some languages, what lists of words to exclude from the index and what rules to use in reducing words to their stems.

These constants are bits in a 64-bit array that consists of two `UInt32` words. In the current implementation the low word is always `0`, so values for the high word are given. If both `UInt32` words are `0`, the default value of `kDefaultLanguagesHighWord` is used.

Phase Values

Define values that are passed to a progress callback function to indicate what phase of an operation Find By Content is currently performing; most of these values are not used in Mac OS X.

```
enum {
    kFBCphIndexing = 0,
    kFBCphFlushing = 1,
    kFBCphMerging = 2,
    kFBCphMakingIndexAccessor = 3,
    kFBCphCompacting = 4,
    kFBCphIndexWaiting = 5,
    kFBCphSearching = 6,
    kFBCphMakingAccessAccessor = 7,
    kFBCphAccessWaiting = 8,
    kFBCphSummarizing = 9,
    kFBCphIdle = 10,
    kFBCphCanceling = 11
};
```

Constants

`kFBCphIndexing`
Indicates an indexing phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphFlushing`
Indicates an indexing phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphMerging`
Indicates an indexing phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphMakingIndexAccessor`
Indicates an indexing phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphCompacting`
Indicates an indexing phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphIndexWaiting`
Indicates an access phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphSearching`
Indicates searching phase. In Mac OS X, this is the only phase value returned to your `FBCCallbackProcPtr` callback.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphMakingAccessAccessor`
Indicates an access phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphAccessWaiting`
Indicates an access phase. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphSummarizing`
Indicates summarization. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphIdle`
Indicates indexing or accessing. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

`kFBCphCanceling`
Indicates cancellation. This is no longer used in Mac OS X.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `FindByContent.h`.

Discussion
The values are meaningless in Mac OS X.

Deprecated Language Constants

Redefine old language constants as new ones; you should use the new value.

```
enum {
    englishHighWord = kFBCenglishHighWord,
    dutchHighWord = kFBCdutchHighWord,
    germanHighWord = kFBCgermanHighWord,
    swedishHighWord = kFBCswedishHighWord,
    danishHighWord = kFBCdanishHighWord,
    spanishHighWord = kFBCspanishHighWord,
    portugueseHighWord = kFBCportugueseHighWord,
    italianHighWord = kFBCitalianHighWord,
    frenchHighWord = kFBCfrenchHighWord,
    romanHighWord = kFBCromanHighWord,
    icelandicHighWord = kFBCicelandicHighWord,
    hebrewHighWord = kFBChebrewHighWord,
    arabicHighWord = kFBCarabicHighWord,
    centeuroHighWord = kFBCcenteuroHighWord,
    croatianHighWord = kFBCcroatianHighWord,
    turkishHighWord = kFBCturkishHighWord,
    romanianHighWord = kFBCromanianHighWord,
    greekHighWord = kFBCgreekHighWord,
    cyrillicHighWord = kFBCcyrillicHighWord,
    devanagariHighWord = kFBCdevanagariHighWord,
    gujuratiHighWord = kFBCgujuratiHighWord,
    gurmukhiHighWord = kFBCgurmukhiHighWord,
    japaneseHighWord = kFBCjapaneseHighWord,
    koreanHighWord = kFBCkoreanHighWord,
    kDefaultLanguagesHighWord = kFBCdefaultLanguagesHighWord
};
```

Result Codes

The result codes returned by Find By Content are listed below.

Result Code	Value	Description
errIANoErr	0	Available in Mac OS X v10.0 and later.
kFBCvTwinExceptionErr	-30500	No telling what it was Available in Mac OS X v10.0 and later.
kFBCnoIndexesFound	-30501	Indexes not found Available in Mac OS X v10.0 and later.
kFBCallocFailed	-30502	Possibly due to low memory Available in Mac OS X v10.0 and later.
kFBCbadParam	-30503	Bad parameter passed to a function Available in Mac OS X v10.0 and later.
kFBCfileNotIndexed	-30504	File not indexed Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kFBCbadIndexFile	-30505	Bad FSSpec, or bad data in file Available in Mac OS X v10.0 and later.
kFBCcompactionFailed	-30506	V-Twin exception caught Available in Mac OS X v10.0 and later.
kFBCvalidationFailed	-30507	V-Twin exception caught Available in Mac OS X v10.0 and later.
kFBCindexingFailed	-30508	V-Twin exception caught Available in Mac OS X v10.0 and later.
kFBCcommitFailed	-30509	V-Twin exception caught Available in Mac OS X v10.0 and later.
kFBCdeletionFailed	-30510	V-Twin exception caught Available in Mac OS X v10.0 and later.
kFBCmoveFailed	-30511	V-Twin exception caught Available in Mac OS X v10.0 and later.
kFBCtokenizationFailed	-30512	Couldn't read from document or query Available in Mac OS X v10.0 and later.
kFBCmergingFailed	-30513	Couldn't merge index files Available in Mac OS X v10.0 and later.
kFBCindexCreationFailed	-30514	Couldn't create index Available in Mac OS X v10.0 and later.
kFBCaccessorStoreFailed	-30515	Available in Mac OS X v10.0 and later.
kFBCaddDocFailed	-30516	Available in Mac OS X v10.0 and later.
kFBCflushFailed	-30517	Available in Mac OS X v10.0 and later.
kFBCindexNotFound	-30518	Index not found Available in Mac OS X v10.0 and later.
kFBCnoSearchSession	-30519	Search session object not created Available in Mac OS X v10.0 and later.
kFBCindexingCanceled	-30520	Indexing cancelled Available in Mac OS X v10.0 and later.
kFBCaccessCanceled	-30521	Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kFBCindexFileDestroyed	-30522	Index file destroyed Available in Mac OS X v10.0 and later.
kFBCindexNotAvailable	-30523	Index not available Available in Mac OS X v10.0 and later.
kFBCsearchFailed	-30524	Search failed Available in Mac OS X v10.0 and later.
kFBCsomeFilesNotIndexed	-30525	Some files are not indexed Available in Mac OS X v10.0 and later.
kFBCillegalSessionChange	-30526	Tried to add/remove volumes to a session. Available in Mac OS X v10.0 and later.
kFBCanalysisNotAvailable	-30527	Available in Mac OS X v10.0 and later.
kFBCbadIndexFileVersion	-30528	Available in Mac OS X v10.0 and later.
kFBCsummarizationCanceled	-30529	Summarization operation cancelled Available in Mac OS X v10.0 and later.
kFBCindexDiskIOFailed	-30530	Available in Mac OS X v10.0 and later.
kFBCbadSearchSession	-30531	Available in Mac OS X v10.0 and later.
kFBCnoSuchHit	-30532	Hit doesn't exist Available in Mac OS X v10.0 and later.
kFBCsummarizationFailed	-30533	Summarization operation failed Available in Mac OS X v10.2 through Mac OS X v10.4.
kFBCnotAllFoldersSearchable	-30533	Not all folders are searchable Available in Mac OS X v10.2 through Mac OS X v10.4.

FontSync Reference

Framework: ApplicationServices/ApplicationServices.h
Declared in FontSync.h

Overview

FontSync is an API that provides a way for your application to identify fonts based upon the content of the font, rather than just the font name. Your application can use FontSync to compare fonts that are available on different computers. This document is relevant for anyone who is writing a text-intensive application that must minimize font mismatch errors when a file is moved from one computer to another. To use this document, you should understand the basics of fonts and be familiar with FontSync references and profiles.

Carbon supports FontSync. However, in Mac OS X, Apple recommends that you use FontSync only in OS X version 10.1 and later.

Functions by Task

Determining Availability, Version, and Feature Information

`FNSEnabled` (page 2427)
Indicates whether FontSync is enabled.

`FNSMatchDefaultsGet` (page 2427)
Determines the default match options used by FontSync functions performing font matching.

`FNSSysInfoGet` (page 2455)
Determines version and feature information for the version of FontSync installed on the user's system.

Providing User Interface Support

`FNSReferenceCountNames` (page 2441)
Determines the number of internal font names in a reference.

`FNSReferenceFindName` (page 2444)
Finds the first name that matches the given font name parameters, if any.

`FNSReferenceGetFamilyInfo` (page 2447)
Obtains information about a font family represented by a font reference.

`FNSReferenceGetIndName` (page 2448)
Finds the font name string and other font name parameters for an indexed font name.

Searching by Font Reference

`FNSProfileMatchReference` (page 2435)
Obtains a list of the references in a profile that match a given reference.

`FNSReferenceMatchFamilies` (page 2452)
Obtains a list of font families that match a reference.

`FNSReferenceMatchFonts` (page 2453)
Obtains a list of font objects that match a reference.

Working With FontSync Profiles

`FNSProfileAddReference` (page 2428)
Adds a font reference to a profile.

`FNSProfileClear` (page 2429)
Removes all font references from a profile.

`FNSProfileClose` (page 2430)
Closes the file associated with a font profile and disposes of run-time data.

`FNSProfileCompact` (page 2430)
Compacts a font profile.

`FNSProfileCountReferences` (page 2431)
Determines the number of font references in a font profile.

`FNSProfileCreate` (page 2432)
Creates an empty FontSync profile using an `FSSpec`.

`FNSProfileCreateWithFSRef` (page 2433)
Creates an empty FontSync profile using an `FSRef`.

`FNSProfileGetIndReference` (page 2434)
Retrieves an indexed font reference from a profile.

`FNSProfileGetVersion` (page 2435)
Retrieves the format version of an open font profile.

`FNSProfileOpen` (page 2437)
Opens an existing font profile using an `FSSpec`.

`FNSProfileOpenWithFSRef` (page 2438)
Opens an existing font profile using an `FSRef`.

`FNSProfileRemoveIndReference` (page 2439)
Deletes an indexed font reference from a profile.

`FNSProfileRemoveReference` (page 2440)
Deletes a font reference from a profile.

Working With FontSync References

`FNSReferenceCreate` (page 2441)
Creates a font reference based on a font object.

Functions

FNSEnabled

Indicates whether FontSync is enabled.

```
Boolean FNSEnabled (
   void
);
```

Return Value
A `Boolean` value indicating whether FontSync is enabled. If `true`, your application can perform FontSync operations. See the Mac Types documentation for a description of the `Boolean` data type.

Discussion
You should check the flag returned by the `FNSEnabled` function before starting a sequence of FontSync calls, although it has no effect on the operation of the rest of the FontSync API.

Version Notes
Available beginning with FontSync 1.0. In FontSync 1.0, `FNSEnabled` always returns `true`.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSMatchDefaultsGet

Determines the default match options used by FontSync functions performing font matching.

```
FNSMatchOptions FNSMatchDefaultsGet (
   void
);
```

Return Value
A bit mask indicating the default match options. This is the value used when the mask constant `kFNSMatchDefaults` is passed to FontSync functions that perform matching. See the description of the `FNSMatchOptions` data type.

Discussion
The `FNSMatchDefaultsGet` function retrieves the bit mask used when the mask constant `kFNSMatchDefaults` is passed to the functions `FNSReferenceMatch` (page 2451), `FNSProfileMatchReference` (page 2435), `FNSReferenceMatchFonts` (page 2453), and `FNSReferenceMatchFamilies` (page 2452). The bit mask value is read from a preferences file which is created when the user sets match criteria via the control panel. The preference file is maintained by the FontSync library. If there is no preferences file, or it is unreadable, the implementation-defined fallback value `kFNSMatchAll` is returned.

There is no API for setting the default match criteria. Your application can specify options that are different from the user's preferences via the bitmask in the FontSync matching calls.

Version Notes
Available beginning with FontSync 1.0. In FontSync 1.0, the implementation-defined fallback value is `kFNSMatchAll`, that is, all defined options turned on. In other words, if the user does not set the match criteria via the control panel, FontSync uses the implementation-defined fallback value of all match criteria selected.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileAddReference

Adds a font reference to a profile.

```
OSStatus FNSProfileAddReference (
   FNSFontProfile iProfile,
   FNSFontReference iReference
);
```

Parameters

`iProfile`
A reference to the font profile to which you want to add a font reference. The profile must be writable.

`iReference`
A reference to the font reference you wish to add.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr`

indicates that a profile does not have a valid structure. The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported format version. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSDuplicateReferenceErr` indicates that an identical reference already exists in the profile. In this case, the new one is not added. The File Manager error `permErr` indicates that the file is either locked and not editable or opened for read-only access. `FNSProfileAddReference` may return other File Manager errors. Memory Manager errors indicate that you did not have enough memory available in your heap.

Discussion
The `FNSProfileAddReference` function adds a font reference to a profile that has read/write access. If an identical reference already exists in the profile, the reference is not added and the result code `kFNSDuplicateReferenceErr` is returned. A matching reference is not necessarily identical, since not all the data in a font reference is examined when a matching operation is performed.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileClear

Removes all font references from a profile.

```
OSStatus FNSProfileClear (
   FNSFontProfile iProfile
);
```

Parameters

iProfile

A font profile reference. Pass a reference to the font profile whose references you wish to remove. The profile must be editable (that is, opened with read/write access).

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that the font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that the profile does not have a valid structure. The File Manager error `permErr` indicates that the file is either locked and not editable or opened for read-only access. `FNSProfileClear` may return other File Manager errors.

Discussion
The `FNSProfileClear` function clears all font references from a specified profile. Note that this is only true for editable profiles (that is, those opened with read/write access). The file of the font profile remains the same size after this operation.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileClose

Closes the file associated with a font profile and disposes of run-time data.

```
OSStatus FNSProfileClose (
   FNSFontProfile iProfile
);
```

Parameters

`iProfile`
A pointer to a font profile reference. Pass a reference to the font profile you wish to close.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure. `FNSProfileClose` may return File Manager errors.

Discussion
The `FNSProfileClose` function closes the file associated with a font profile. Any memory associated with the reference is released. You should call the function `FNSProfileCompact` (page 2430) before closing a profile that has been edited, since closing a profile does not automatically compact it.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileCompact

Compacts a font profile.

```
OSStatus FNSProfileCompact (
   FNSFontProfile iProfile
);
```

Parameters

iProfile

A font profile reference. Pass a reference to the font profile you wish to compact. The profile must be editable (that is, opened with read/write access).

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure. `FNSProfileCompact` may return File Manager errors.

Discussion

The `FNSProfileCompact` function eliminates excess space created when creating a font profile (that is, the space you designate for not-yet-existent font references). This space is necessary to minimize growing the file and shuffling data. If a profile has not been opened for read/write access, `FNSProfileCompact` simply returns without doing anything.

You should call `FNSProfileCompact` before closing a profile that has been edited.

Version Notes

Available beginning with FontSync 1.0.

Availability

Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`FontSync.h`

FNSProfileCountReferences

Determines the number of font references in a font profile.

```
OSStatus FNSProfileCountReferences (
   FNSFontProfile iProfile,
   ItemCount *oCount
);
```

Parameters

iProfile

A reference to the font profile whose font references you wish to count.

oCount

On return, a pointer to the number of font references in the profile.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that a profile is valid, but created by a later version of FontSync, or that a profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileCreate

Creates an empty FontSync profile using an `FSSpec`.

```
OSStatus FNSProfileCreate (
   const FSSpec *iFile,
   FourCharCode iCreator,
   ItemCount iEstNumRefs,
   FNSObjectVersion iDesiredVersion,
   FNSFontProfile *oProfile
);
```

Parameters

iFile
A pointer to the file that you want to initialize as an empty font profile.

iCreator
The creator code to set for the file. To specify the file creator code assigned by FontSync, pass the `kFNSCreatorDefault` constant, described in "Font Profile Constants" (page 2461).

iEstNumRefs
The estimated number of font references that the font profile will contain. Estimating this value minimizes the number of times the file needs to be grown, since the new profile will usually immediately have font references added to it. Pass 0 if you don't know how many font references your profile will contain.

iDesiredVersion
The desired format version of the font profile. Pass a value in the range returned by the function `FNSSysInfoGet` (page 2455) in the `oCurProfileVersion` and `oMinProfileVersion` fields of the system information structure. To specify the most recent version supported by the FontSync library regardless of format version, pass the constant `kFNSVersionDontCare`, described in "Version Constants" (page 2462).

oProfile
On return, a pointer to a reference to the newly-created font profile.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that you requested an unsupported profile format version. Memory Manager errors indicate that the font profile could not be created because you did not have enough memory available in your heap. `FNSProfileCreate` may return File Manager errors.

Discussion
The `FNSProfileCreate` function creates an empty file containing a FontSync font profile. The newly-created font profile is ready for use. You can add font references to the profile by calling the function `FNSProfileAddReference` (page 2428).

`FNSProfileCreate` requires that you specify the desired profile version format because there will likely be changes to the profile file format in future versions. This allows earlier versions of FontSync to use the font profiles you create.

Version Notes
Available beginning with FontSync 1.0. In FontSync 1.0, if you specify the constant `kFNSCreatorDefault` in the `iCreator` parameter of the function `FNSProfileCreate`, FontSync assigns the creator code `'fns'`

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileCreateWithFSRef

Creates an empty FontSync profile using an `FSRef`.

```
OSStatus FNSProfileCreateWithFSRef (
   const FSRef *iParentDirectory,
   UniCharCount iNameLength,
   const UniChar *iName,
   FourCharCode iCreator,
   ItemCount iEstNumRefs,
   FNSObjectVersion iDesiredVersion,
   FNSFontProfile *oProfile
);
```

Parameters

iParentDirectory
A pointer to the parent directory of the file that you want to initialize as an empty font profile.

iNameLength
The number of `UniChar` characters in the `iName` parameter.

iName
The name of the file in which you are storing the profile.

iCreator
The creator code to set for the file. To specify the file creator code assigned by FontSync, pass the `kFNSCreatorDefault` constant, described in "Font Profile Constants" (page 2461).

iEstNumRefs
The estimated number of font references that the font profile will contain. Estimating this value minimizes the number of times the file needs to be grown, since the new profile will usually immediately have font references added to it. Pass 0 if you don't know how many font references your profile will contain.

iDesiredVersion

The desired format version of the font profile. Pass a value in the range returned by the function `FNSSysInfoGet` (page 2455) in the `oCurProfileVersion` and `oMinProfileVersion` fields of the system information structure. To specify the most recent version supported by the FontSync library regardless of format version, pass the constant `kFNSVersionDontCare`, described in "Version Constants" (page 2462).

oProfile

On return, a pointer to the newly-created font profile.

Return Value
A result code. See "FontSync Result Codes" (page 2462).

Discussion
The function `FNSProfileCreateWithFSRef` works similarly to the function `FNSProfileCreate`, except that `FNSProfileCreateWithFSRef` uses an `FSRef` instead of an `FSSpec`. An `FSSpec` cannot handle Unicode names that are too long, as long names are truncated. In addition, an `FSSpec` cannot be shared between processes since an `FSSpec` references volume IDs which are different between different processes.

Availability
Not available in CarbonLib 1.0.
Available in Mac OS X 10.1 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileGetIndReference

Retrieves an indexed font reference from a profile.

```
OSStatus FNSProfileGetIndReference (
   FNSFontProfile iProfile,
   UInt32 iWhichReference,
   FNSFontReference *oReference
);
```

Parameters

iProfile

A reference to the font profile whose indexed font reference you want to determine.

iWhichReference

An index into the list of font references in the profile. Pass a value between 0 and one less than the number of references in the profile returned by the function `FNSProfileCountReferences` (page 2431).

oReference

On return, a pointer to a reference to the indexed font reference.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that the font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `inputOutOfBounds` indicates that the specified index was out of range. The result code `kFNSInvalidProfileErr` indicates that the profile does not have a valid structure. `FNSProfileGetIndReference` may return File Manager errors. Memory Manager errors indicate that you did not have enough memory available in your heap.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileGetVersion

Retrieves the format version of an open font profile.

```
OSStatus FNSProfileGetVersion (
   FNSFontProfile iProfile,
   FNSObjectVersion *oVersion
);
```

Parameters

iProfile
A reference to the font profile whose format version number you wish to obtain.

oVersion
On return, a pointer to the format version number of the font profile.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSInvalidProfileErr` indicates that the profile does not have a valid structure.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileMatchReference

Obtains a list of the references in a profile that match a given reference.

```
OSStatus FNSProfileMatchReference (
   FNSFontProfile iProfile,
   FNSFontReference iReference,
   FNSMatchOptions iMatchOptions,
   ItemCount iOutputSize,
   UInt32 oIndices[],
   ItemCount *oNumMatches
);
```

Parameters

iProfile

A reference to the font profile containing the font references you wish to compare.

iReference

A reference to a font reference against which you are performing the comparison.

iMatchOptions

A bit mask you can use to set the matching option bits to be used in the comparison. To specify the global default match criteria, pass the bit mask returned by the function `FNSMatchDefaultsGet` (page 2427). Your application can specify options that are different from the user's preferences via this mask.

iOutputSize

The number of font references you want passed back in the `oIndices` array. This may be less than the actual number of matches passed back in the `oNumMatches` parameter. To determine this value, see the discussion below.

oIndices

On return, a pointer to an array of indices identifying the font references that matched. The number of indices returned is limited by the value you specify in the `iOutputSize` parameter. The total number of matching references is passed back in the `oNumMatches` parameter.

oNumMatches

On return, a pointer to the total number of matching font references. This value may be greater than the number of indices passed back in the `oIndices` array.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported version number. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure. The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSMismatchErr` indicates that no matches were found. The File Manager error `permErr` indicates that the file is either locked and not editable or opened for read-only access. `FNSProfileMatchReference` may return other File Manager errors. Memory Manager errors indicate that you did not have enough memory available in your heap.

Discussion

The `FNSProfileMatchReference` function obtains a list of the font references that match a specified reference. Since there may be more than one matching reference, a list is returned.

The number of font references passed back in the `oIndices` array is limited by the value you specify in the `iOutputSize` parameter. The actual number of matches is passed back in the `oNumMatches` parameter. You can check this value to determine whether the `oIndices` array was large enough to contain the matches.

If you want to determine whether the profile has a matching font, but don't care which one, pass 0 for the `iOutputSize` parameter and `NULL` for the `oNumMatches` parameter. The result code `noErr` indicates that matches were found, while the result code `kFNSMismatchErr` indicates that no matches were found.

To determine the number of matches, call `FNSProfileMatchReference` and pass 0 for the `iOutputSize` parameter. The pointer passed back in the `oNumMatches` parameter will point to the actual number of matches. You can then call `FNSProfileMatchReference` again, passing the returned number of matches in the `iOutputSize` parameter.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileOpen

Opens an existing font profile using an `FSSpec`.

```
OSStatus FNSProfileOpen (
   const FSSpec *iFile,
   Boolean iOpenForWrite,
   FNSFontProfile *oProfile
);
```

Parameters

iFile
A pointer to the font profile file that you wish to open.

iOpenForWrite
A flag indicating whether the profile file is read/write or read-only. Pass `true` to allow read/write access. This is necessary if the profile is going to be editable.

oProfile
On return, a pointer to a reference to the open font profile.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure. Memory Manager errors indicate that you did not have enough memory available in your heap. `FNSProfileOpen` may return File Manager errors.

Discussion
The `FNSProfileOpen` function opens an already-existing font profile (that is, one that contains font references). If you want to make the font profile editable, pass true in the `iOpenForWrite` parameter. `FNSProfileOpen` will not open an empty profile created by the function `FNSProfileCreate` (page 2432).

Font profiles are housed in a file. FontSync attempts to moderate access to this file. Ideally, it tries to either allow many readers or exactly one writer but not both. The Mac OS File Manager does not allow this kind of exclusion on local volumes, so it may still be possible for someone to get write access to a profile when there are active readers. Rather than complicating the implementation to work around this limitation, FontSync profile files are treated like most document files. That is, the caller is responsible for making sure this does not occur. If the user wishes to modify a profile, your application should make a copy of the file, modify the copy, and swap file names when done. This has the added benefit of preserving the original profile if an error leaves the new profile invalid.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileOpenWithFSRef

Opens an existing font profile using an `FSRef`.

```
OSStatus FNSProfileOpenWithFSRef (
   const FSRef *iFile,
   Boolean iOpenForWrite,
   FNSFontProfile *oProfile
);
```

Parameters

`iFile`
A pointer to the `FSRef` that specifies the file that you wish to open.

`iOpenForWrite`
A flag indicating whether the profile file is read/write or read-only. Pass `true` to allow read/write access. This is necessary if the profile is going to be editable.

`oProfile`
On return, a pointer to a reference to the open font profile.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure. Memory Manager errors indicate that you did not have enough memory available in your heap. `FNSProfileOpen` may return File Manager errors.

Discussion
The function `FNSProfileOpenWithFSRef` works similarly to the function `FNSProfileOpen`, `FNSProfileOpenWithFSRef` uses an `FSRef` instead of an `FSSpec`. An `FSSpec` cannot handle Unicode names that are too long, as long names are truncated. In addition, an `FSSpec` cannot be shared between processes since an `FSSpec` references volume IDs which are different between different processes.

Availability
Not available in CarbonLib 1.0.
Available in Mac OS X 10.1 and later
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileRemoveIndReference

Deletes an indexed font reference from a profile.

```
OSStatus FNSProfileRemoveIndReference (
   FNSFontProfile iProfile,
   UInt32 iIndex
);
```

Parameters

iProfile

A reference to the font profile whose indexed font reference you want to delete. The profile must be writable.

iIndex

An index into the list of font references in the profile. Pass a value between 0 and one less than the number of references in the profile, returned by the function `FNSProfileCountReferences` (page 2431). Note that this will change the indices of all succeeding references.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure. The result code `permErr` indicates that a specified file is locked and not writable. The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported format version. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `inputOutOfBounds` indicates that the specified index was out of range. The File Manager error `permErr` indicates that the file is either locked and not editable or opened for read-only access. `FNSProfileRemoveIndReference` may return other File Manager errors. Memory Manager errors indicate that you did not have enough memory available in your heap.

Discussion
The `FNSProfileRemoveIndReference` function deletes an indexed font reference from an editable profile. The reference must be identical to the reference specified in the `iReference` parameter. A matching reference is not enough, since not all the data in a font reference is examined when a matching operation is performed.

You can use either `FNSProfileRemoveIndReference` or the function `FNSProfileRemoveReference` (page 2440) to remove a font reference, depending upon what you know about the reference. If you know its value, call `FNSProfileRemoveReference` (page 2440). If you know its index in the list of font references, call `FNSProfileRemoveIndReference`.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSProfileRemoveReference

Deletes a font reference from a profile.

```
OSStatus FNSProfileRemoveReference (
   FNSFontProfile iProfile,
   FNSFontReference iReference
);
```

Parameters

iProfile

A reference to the font profile whose font reference you want to delete. The profile must be writable.

iReference

A reference to the font reference you wish to remove.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadProfileVersionErr` indicates that a font profile has an unsupported format version. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. The result code `kFNSInvalidProfileErr` indicates that a profile does not have a valid structure. The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported format version. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSMismatchErr` indicates that the reference you wish to remove is not in the profile. The File Manager error `permErr` indicates that the file is either locked and not editable or opened for read-only access. `FNSProfileRemoveReference` may return other File Manager errors. Memory Manager errors indicate that you did not have enough memory available in your heap.

Discussion
The `FNSProfileRemoveReference` function deletes a font reference from an editable profile. The reference must be identical to the reference specified in the `iReference` parameter. A matching reference is not necessarily identical, since not all the data in a font reference is examined when a matching operation is performed.

You can use either `FNSProfileRemoveReference` or the function `FNSProfileRemoveIndReference` (page 2439) to remove a font reference, depending upon what you know about the reference. If you know the value of the reference, call `FNSProfileRemoveReference`. If you know its index in the list of font references, call `FNSProfileRemoveIndReference` (page 2439).

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceCountNames

Determines the number of internal font names in a reference.

```
OSStatus FNSReferenceCountNames (
   FNSFontReference iReference,
   ItemCount *oNameCount
);
```

Parameters

`iReference`

A reference to the font reference whose font names you wish to count.

`oNameCount`

On return, a pointer to the number of internal font names, other than the font family name passed to the `GetFNum` function, recorded in the reference. The font family passed to the function `GetFNum` is available by calling the function `GetFamilyInfo`. This includes the PostScript and unique names, if available.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSInsufficientDataErr` indicates that the mask constant `kFNSMissingDataNoMatch` was set and both references being compared are missing the same data. The result code `kFNSMismatchErr` indicates that no font names were recorded in the reference.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceCreate

Creates a font reference based on a font object.

```
OSStatus FNSReferenceCreate (
   FMFont iFont,
   FNSObjectVersion iDesiredVersion,
   FNSFontReference *oReference
);
```

Parameters

iFont

The font object ID representing the font whose reference you wish to create.

iDesiredVersion

The desired font reference format version number. Pass a value between the oldest and current format version numbers supported by the FontSync library. You can determine this range by examining the `oCurRefVersion` and `oMinRefVersion` fields of the `FNSSysInfo` (page 2458) structure. This structure is passed back in the `ioInfo` parameter of the function `FNSSysInfoGet` (page 2455). To specify the most recent version supported by the FontSync library regardless of format version, pass the constant `kFNSVersionDontCare`, described in "Version Constants" (page 2462).

oReference

On return, a pointer to a reference to the newly-created FontSync reference.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that you requested an unsupported reference format version. The Font Manager result code `kFMInvalidFontErr` indicates that a font is invalid. Memory Manager errors indicate that a font reference could not be created because you did not have enough memory available in your heap.

Discussion

You should call the `FNSReferenceCreate` function to create a font reference if your application uses ATSUI to render text. If the specified font object is associated with a font family, the newly-created font reference will contain the QuickDraw Text-specific information from that associated family.

The `FNSReferenceCreate` function requires that you specify the desired font reference format version because there will likely be changes to the nature of the "fingerprints" in a font reference in future versions. This allows earlier versions of FontSync to use the font references you create.

Version Notes

Available beginning with FontSync 1.0. In FontSync 1.0, a font object can only belong to one family. FontSync uses the family returned by the Font Manager function `FMGetFontFamilyInstanceFromFont`.

Availability

Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`FontSync.h`

FNSReferenceCreateFromFamily

Creates a font reference based on a font family and style.

```
OSStatus FNSReferenceCreateFromFamily (
   FMFontFamily iFamily,
   FMFontStyle iStyle,
   FNSObjectVersion iDesiredVersion,
   FNSFontReference *oReference,
   FMFontStyle *oActualStyle
);
```

Parameters

iFamily

The font family of the font whose reference you wish to create.

iStyle

The style of the font family. This value is often not the actual style of the font reference being created, since there are often left-over style bits. The actual style of the newly-created font reference is passed back in the `oActualStyle` parameter. For more information, see the discussion.

iDesiredVersion

The desired format version of the font reference. Pass a value in the range returned by the function `FNSSysInfoGet` (page 2455) in the `oCurRefVersion` and `oMinRefVersion` fields of the system information structure. To specify the most recent version supported by the FontSync library regardless of format version, pass the constant `kFNSVersionDontCare`, described in "Version Constants" (page 2462).

oReference

On return, a pointer to a reference to the newly-created FontSync reference.

oActualStyle

On return, a pointer to the actual style of the newly-created font reference. This value may differ from the value you passed in the `iStyle` parameter. For more information, see the discussion. This value may be `NULL`.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that you requested an unsupported reference format version. The Font Manager result code `kFMInvalidFontFamilyErr` indicates that a font family is invalid. Memory Manager errors indicate that a font reference could not be created because you did not have enough memory available in your heap.

Discussion

You should call the `FNSReferenceCreateFromFamily` function to create a font reference if your application uses QuickDraw Text to render text.

The style you specify in the `iStyle` parameter is often not the actual style of the font reference being created, since there may not be a real face corresponding to that style. For example, a family may not have a real italic face, so any italicization is handled by skewing the glyphs. The actual style of the newly-created font reference is passed back in the `oActualStyle` parameter.

The `FNSReferenceCreateFromFamily` function requires that you specify the desired font reference format version because there will likely be changes to the nature of the "fingerprints" in a font reference in future versions. This allows earlier versions of FontSync to use the font references you create.

Version Notes

Available beginning with FontSync 1.0.

Availability

Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceDispose

Disposes of the storage associated with a font reference.

```
OSStatus FNSReferenceDispose (
   FNSFontReference iReference
);
```

Parameters

iReference

A pointer to the font reference whose associated memory you wish to dispose of.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported format version. This may indicate that the font reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. Memory Manager errors indicate that you did not have enough memory available in your heap.

Special Considerations

You should not use a font reference after calling the `FNSReferenceDispose` function to dispose of its storage.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceFindName

Finds the first name that matches the given font name parameters, if any.

```
OSStatus FNSReferenceFindName (
   FNSFontReference iReference,
   FontNameCode iFontNameCode,
   FontPlatformCode iFontNamePlatform,
   FontScriptCode iFontNameScript,
   FontLanguageCode iFontNameLanguage,
   ByteCount iMaximumNameLength,
   Ptr oName,
   ByteCount *oActualNameLength,
   ItemCount *oFontNameIndex
);
```

Parameters

iReference

A reference to the font reference whose font name you are searching for.

iFontNameCode

The type of the font name string you are searching for.

iFontNamePlatform

The encoding of the font name string you are searching for. You can pass the `kFontNoPlatform` constant if you do not care about the encoding of a font name. In this case, `FNSReferenceFindName` will pass back the first name matching the other font name parameters.

iFontNameScript

The script code of the font name string you are searching for. You can pass the `kFontNoScript` constant if you do not care about the script ID. In this case, `FNSReferenceFindName` will pass back the first name matching the other font name parameters.

iFontNameLanguage

The language code of the font name string you are searching for. You can pass the `kFontNoLanguage` constant if you do not care about the language of the font name. In this case, `FNSReferenceFindName` will pass back the first name matching the other font name parameters.

iMaximumNameLength

The maximum length of the font name. Typically, this is equivalent to the size of the buffer allocated to contain the font name pointed to by the `oName` parameter. To determine this length, see the discussion below.

oName

A pointer to a buffer. Before calling `FNSReferenceFindName`, pass a pointer to memory that you have allocated for this buffer. On return, the buffer contains the font name string. If the buffer you allocate is not large enough, `FNSReferenceFindName` passes back a partial string.

oActualNameLength

On return, a pointer to the actual length of the font name string. This may be greater than the value passed in the `iMaximumNameLength` parameter. You should check this value to make sure that you allocated enough memory for the buffer.

oFontNameIndex

On return, a pointer to a 0-based index of the font name in the font name table. This can be used with the function `FNSReferenceGetIndName` (page 2448) to determine the actual values of unknown font name parameters.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code

kFNSInsufficientDataErr indicates that the mask constant kFNSMissingDataNoMatch was set and both references being compared are missing the same data. The result code kFNSNameNotFoundErr indicates that there was no name in the font reference that matched the given parameters.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
FontSync.h

FNSReferenceFlatten

Flattens a font reference.

```
OSStatus FNSReferenceFlatten (
   FNSFontReference iReference,
   void *oFlatReference,
   ByteCount *oFlattenedSize
);
```

Parameters

iReference
The font reference that you want to flatten.

oFlatReference
A pointer to the storage for the font reference to be flattened. Pass a NULL pointer if you wish to determine the size of the flattened reference without actually creating it.

oFlattenedSize
On return, a pointer to the flattened size (in bytes) of the font reference.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code kFNSBadReferenceVersionErr indicates that a font reference has an unsupported format version. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code kFNSInvalidReferenceErr indicates that a font reference is invalid.

Discussion
The FNSReferenceFlatten function flattens a font reference into a form which can be stored externally (for example, in a document or embedded in an Apple event), and returns the size of the flattened reference in the oFlattenedSize parameter. FNSReferenceFlatten assumes that the storage pointed to by iFlatReference is large enough to hold the data and will always contain a full flattened reference.

If you simply want to calculate the size of a flattened reference, you can pass a NULL pointer in the iFlatReference parameter or call the function FNSReferenceFlattenedSize (page 2447).

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceFlattenedSize

Calculates the space required for the flattened form of a font reference.

```
OSStatus FNSReferenceFlattenedSize (
   FNSFontReference iReference,
   ByteCount *oFlattenedSize
);
```

Parameters

`iReference`
The font reference whose flattened form you wish to compute.

`oFlattenedSize`
On return, a pointer to the flattened size (in bytes) of the font reference.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that the font reference has an unsupported format version. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that the font reference is invalid.

Discussion
You can call the `FNSReferenceFlattenedSize` function to calculate the size of a flattened reference. You can also accomplish this by passing a `NULL` pointer in the `iFlatReference` parameter of the function `FNSReferenceFlatten` (page 2446).

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceGetFamilyInfo

Obtains information about a font family represented by a font reference.

```
OSStatus FNSReferenceGetFamilyInfo (
   FNSFontReference iReference,
   Str255 oFamilyName,
   ScriptCode *oFamilyNameScript,
   FMFontStyle *oActualStyle
);
```

Parameters

iReference

A reference to the font reference representing a font family.

oFamilyName

On return, the name by which the font is known to the classic Font Manager (that is, the string you pass to the Font Manager function `GetFNum`). If you do not want to obtain this information, pass `NULL`.

oFamilyNameScript

On return, a pointer to the script code of the family name string. If you do not want to obtain this information, pass `NULL`.

oActualStyle

On return, a pointer to the actual QuickDraw style associated with the font reference. This is the value passed back in the `oActualStyle` parameter of the function `FNSReferenceCreateFromFamily` (page 2442). If you do not want to obtain this information, pass `NULL`. For more information, see the discussion of `FNSReferenceCreateFromFamily` (page 2442).

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSInsufficientDataErr` indicates that the mask constant `kFNSMissingDataNoMatch` was set and both references being compared are missing the same data. The result code `kFNSMismatchErr` indicates that no font names were recorded in the reference.

Version Notes

Available beginning with FontSync 1.0.

Availability

Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`FontSync.h`

FNSReferenceGetIndName

Finds the font name string and other font name parameters for an indexed font name.

```
OSStatus FNSReferenceGetIndName (
   FNSFontReference iReference,
   ItemCount iFontNameIndex,
   ByteCount iMaximumNameLength,
   Ptr oName,
   ByteCount *oActualNameLength,
   FontNameCode *oFontNameCode,
   FontPlatformCode *oFontNamePlatform,
   FontScriptCode *oFontNameScript,
   FontLanguageCode *oFontNameLanguage
);
```

Parameters

iReference

A reference to the font reference whose indexed font name you want information about.

iFontNameIndex

An index of the font name you want information about. Pass a value between 0 and one less than the count passed back by the function `FNSProfileCountReferences` (page 2431).

iMaximumNameLength

The maximum length of the font name. Typically, this is equivalent to the size of the buffer allocated to contain the font name pointed to by the `oName` parameter. To determine this length, see the discussion below.

oName

A pointer to a buffer. Before calling `FNSReferenceGetIndName`, pass a pointer to memory that you have allocated for this buffer. If you are uncertain of how much memory to allocate, see the discussion below. On return, the buffer contains the font name string. If the buffer you allocate is not large enough, `FNSReferenceGetIndName` passes back a partial string.

oActualNameLength

On return, a pointer to the actual length of the font name string. This may be greater than the value passed in the `iMaximumNameLength` parameter. You should check this value to make sure that you allocated enough memory for the buffer.

oFontNameCode

On return, a pointer to the type of the font name string.

oFontNamePlatform

On return, a pointer to the encoding of the font name string.

oFontNameScript

On return, a pointer to the script ID of the font name string.

oFontNameLanguage

On return, a pointer to the language of the font name string.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSInsufficientDataErr` indicates that the mask constant `kFNSMissingDataNoMatch` was set and both references being compared are missing the same data. The result code `inputOutOfBounds` indicates that the specified index was out of range.

Discussion
You should call the `FNSReferenceGetIndName` function to iterate through the entries of a font name table to find the font name string, name code, language code, script code, and platform code of an indexed font name.

The best way to use `FNSReferenceGetIndName` is to call it twice:

- Pass the reference of the font whose name table you want to iterate in the `iReference` parameter, `NULL` for the `oName` parameter, and 0 for the other parameters. `FNSReferenceGetIndName` returns the length of the font name string in the `oActualNameLength` parameter.
- Allocate enough space for a font name buffer of the returned size, then call the function again, passing a pointer in the `oName` parameter; on return, the pointer references the font name string.

To find the index and font name of the first font in a name table matching given font name parameters, call the function `FNSReferenceFindName` (page 2444).

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceGetVersion

Indicates the format version number of a font reference.

```
OSStatus FNSReferenceGetVersion (
   FNSFontReference iReference,
   FNSObjectVersion *oVersion
);
```

Parameters

iReference
The font reference whose format version number you wish to determine.

oVersion
On return, a pointer to the format version number of the specified font reference.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSInvalidReferenceErr` indicates that the font reference is invalid.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceMatch

Compares font references using specified matching options.

```
OSStatus FNSReferenceMatch (
   FNSFontReference iReference1,
   FNSFontReference iReference2,
   FNSMatchOptions iOptions,
   FNSMatchOptions *oFailedMatchOptions
);
```

Parameters

iReference1

A font reference whose contents you wish to compare to the font reference in the `iReference2` parameter.

iReference2

A font reference whose contents you wish to compare to the font reference in the `iReference1`parameter.

iOptions

A bit mask you can use to set the match option bits to be used in the font comparison. To specify the global default match criteria, pass the bit mask returned by the function `FNSMatchDefaultsGet` (page 2427). Your application can specify options that are different from the user's preferences via this mask.

oFailedMatchOptions

Before calling `FNSReferenceMatch`, pass `NULL` if you do not desire to know which match options failed. On return, a pointer to a bit mask that you can test to determine the match options that failed to match in the event of a mismatch.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported format version. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSMismatchErr` indicates that a font reference did not match. The result code `kFNSInsufficientDataErr` indicates that the mask constant `kFNSMissingDataNoMatch` was set and both references being compared are missing the same data.

Discussion
The `FNSReferenceMatch` function returns a bit mask indicating the matching options that did not match when comparing two font references. You should specify which match options you wish to compare in the `iOptions` parameter. To specify the default match criteria, pass the bit mask returned by the function `FNSMatchDefaultsGet` (page 2427). If the match fails, on return, the `oFailedMatchOptions` parameter contains a bit mask of the elements that failed to match. You can use the bit mask to determine the criteria under which the fonts failed to match.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
FontSync.h

FNSReferenceMatchFamilies

Obtains a list of font families that match a reference.

```
OSStatus FNSReferenceMatchFamilies (
   FNSFontReference iReference,
   FNSMatchOptions iMatchOptions,
   ItemCount iOutputSize,
   FMFontFamilyInstance oFonts[],
   ItemCount *oNumMatches
);
```

Parameters

iReference

A reference to the font reference whose matching font(s) you wish to determine.

iMatchOptions

A bit mask you can use to set the matching option bits to be used in the comparison. To specify the global default match criteria, pass the bit mask returned by the function FNSMatchDefaultsGet (page 2427). The total number of matching references is passed back in the oNumMatches parameter. Your application can specify options that are different from the user's preferences via this mask.

iOutputSize

The capacity of the oFonts array. This may be less than the actual number of matches passed back in the oNumMatches parameter.

oFonts

On return, a pointer to an array of indices identifying the fonts matching the specified reference. The number of indices returned is limited by the value you specify in the iOutputSize parameter.

oNumMatches

On return, a pointer to the total number of font families that match the specified reference. This value may be greater than the number of fonts passed back in the oFonts array.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code kFNSBadReferenceVersionErr indicates that a font reference has an unsupported version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code kFNSInvalidReferenceErr indicates that a font reference is invalid. The result code kFNSMismatchErr indicates that no matches were found. Memory Manager errors indicate that you did not have enough memory available in your heap.

Discussion

The FNSReferenceMatchFamilies function maps a font reference to an active font that can be used with QuickDraw Text. Since there may be more than one such font, a list is returned.

The number of fonts passed back in the oFonts array is limited by the value you specify in the iOutputSize parameter. The actual number of matches is passed back in the oNumMatches parameter. You can check this value to determine whether the oFonts array was large enough to contain the matches.

If FNSReferenceMatchFamilies cannot find a font family that matches a font reference and someone has registered interest in this process, FontSync sends an Apple Event with the details of the request to the third party font-management utility in question. For more information, see the discussion for the function FNSReferenceMatchFonts (page 2453).

If you want to determine whether the profile has a matching font, but don't care which one, pass 0 for the iOutputSize parameter and NULL for the oNumMatches parameter. The result code noErr indicates that matches were found, while the result code kFNSMismatchErr indicates that no matches were found.

To determine the number of matches, call FNSReferenceMatchFamilies and pass 0 for the iOutputSize parameter. The pointer passed back in the oNumMatches parameter will point to the actual number of matches. You can then call FNSReferenceMatchFamilies again, passing the returned number of matches in the iOutputSize parameter.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
FontSync.h

FNSReferenceMatchFonts

Obtains a list of font objects that match a reference.

```
OSStatus FNSReferenceMatchFonts (
   FNSFontReference iReference,
   FNSMatchOptions iMatchOptions,
   ItemCount iOutputSize,
   FMFont oFonts[],
   ItemCount *oNumMatches
);
```

Parameters

iReference
A reference to the font reference whose matching font(s) you wish to determine.

iMatchOptions
A bit mask you can use to set the matching option bits to be used in the comparison. To specify the global default match criteria, pass the bit mask returned by the function FNSMatchDefaultsGet (page 2427). The total number of matching references is passed back in the oNumMatches parameter. Your application can specify options that are different from the user's preferences via this mask.

iOutputSize
The capacity of the oFonts array. This may be less than the actual number of matches passed back in the oNumMatches parameter.

oFonts
On return, a pointer to an array of indices identifying the fonts matching the specified reference. The number of indices returned is limited by the value you specify in the iOutputSize parameter.

oNumMatches

On return, a pointer to the total number of font objects that match the specified reference. This value may be greater than the number of fonts passed back in the `oFonts` array.

Return Value
A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported format version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSInvalidReferenceErr` indicates that a font reference is invalid. The result code `kFNSMismatchErr` indicates that no matches were found. Memory Manager errors indicate that you did not have enough memory available in your heap.

Discussion
The `FNSReferenceMatchFonts` function passes back a list of active fonts which match the specified reference. `FNSReferenceMatchFonts` maps a font reference to an actual font that can be used with ATSUI. Since there may be more than one such font, a list is returned.

The number of fonts passed back in the `oFonts` array is limited by the value you specify in the `iOutputSize` parameter. The actual number of matches is passed back in the `oNumMatches` parameter. You can check this value to determine whether the `oFonts` array was large enough to contain the matches.

If you want to determine whether the profile has a matching font, but don't care which one, pass 0 for the `iOutputSize` parameter and `NULL` for the `oNumMatches` parameter. The result code `noErr` indicates that matches were found, while the result code `kFNSMismatchErr` indicates that no matches were found.

To determine the number of matches, call `FNSReferenceMatchFonts` and pass 0 for the `iOutputSize` parameter. The pointer passed back in the `oNumMatches` parameter will point to the actual number of matches. You can then call `FNSReferenceMatchFonts` again, passing the returned number of matches in the `iOutputSize` parameter.

If `FNSReferenceMatchFonts` cannot find an active font that matches a font reference and your application has registered interest in this process, FontSync sends an Apple Event with the details of the request to the third party font-management utility in question. The receiver should respond with a list of matching fonts, taking whatever steps are necessary to identify and activate them before replying to the event. Registration is handled by the simple expedient of installing a handler for the appropriate Apple Event. This handler will typically be installed in the system table, though FontSync will check for handlers both in the system and in the context's local handler table. The Apple Event will be a send-to-self.

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`FontSync.h`

FNSReferenceUnflatten

Reconstitutes a flattened font reference.

```
OSStatus FNSReferenceUnflatten (
   const void *iFlatReference,
   ByteCount iFlattenedSize,
   FNSFontReference *oReference
);
```

Parameters

iFlatReference

A pointer to the flattened font reference.

iFlattenedSize

The size (in bytes) of the flattened font reference.

oReference

On return, a pointer to a reference to the reconstituted font reference.

Return Value

A result code. See "FontSync Result Codes" (page 2462). The result code `kFNSBadReferenceVersionErr` indicates that a font reference has an unsupported format version. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid. The result code `kFNSBadFlattenedSizeErr` indicates that either the specified size doesn't match the size recorded in the flattened reference or the size was not large enough to hold a flattened reference. The result code `kFNSInvalidReferenceErr` indicates that a reconstructed reference is bad. Memory Manager errors indicate that you did not have enough memory available in your heap.

Discussion

The `FNSReferenceUnflatten` function reconstitutes a flattened font reference from its external form. For example, you could use `FNSReferenceUnflatten` to read a font reference out of a document. The `iFlattenedSize` parameter is not really necessary since a flattened reference contains its own size. However, you can use this value to check that you have passed the right amount of data for the flattened reference.

Version Notes

Available beginning with FontSync 1.0.

Availability

Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`FontSync.h`

FNSSysInfoGet

Determines version and feature information for the version of FontSync installed on the user's system.

```
void FNSSysInfoGet (
   FNSSysInfo *ioInfo
);
```

Parameters

ioInfo

Before calling the `FNSSysInfoGet` function, pass a pointer to a `FNSSysInfo` (page 2458) structure. Fill in the `iSysInfoVersion` field of the structure with the version of this structure. Pass the constant `kFNSCurSysInfoVersion`, described in "Version Constants" (page 2462), to represent the current version. On return, `FNSSysInfoGet` fills in the remaining fields and passes back a pointer to the structure.

Discussion

Before calling the `FNSSysInfoGet` function, you should fill in the `iSysInfoVersion` field of the `FNSSysInfo` (page 2458) structure with the version of this structure. Pass the constant `kFNSCurSysInfoVersion`, described in "Version Constants" (page 2462) , to represent the current version. `FNSSysInfoGet` fills in the remaining fields and passes back the structure in the `ioInfo` parameter. The information it provides includes the version of FontSync running in the current context and available features, as well as the current and oldest font reference and profile format versions supported by the FontSync library.

New fields may be added to the end of the structure in future versions of FontSync. FontSync uses the `iSysInfoVersion` field to determine which version of the structure you are using. The value of the current version constant `kFNSCurSysInfoVersion` will change accordingly.

Version Notes

Available beginning with FontSync 1.0. In FontSync 1.0, the current structure version is defined by the constant `kFNSCurSysInfoVersion`, described in "Version Constants" (page 2462).

Availability

Available in CarbonLib 1.0 and later when Font Sync 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`FontSync.h`

Data Types

FNSFeatureFlags

Represents a mask that you can use to determine available FontSync features.

```
typedef UInt32 FNSFeatureFlags;
```

Discussion

The `FNSFeatureFlags` type defines a bit mask your application can use to determine available FontSync features. The function `FNSSysInfoGet` (page 2455) passes back a mask of this type in the `oFeatures` field of the `FNSSysInfo` (page 2458) structure in the `ioInfo` parameter. You can use this mask to determine available FontSync features.

Version Notes

Available beginning with FontSync 1.0. In FontSync 1.0, the value is 0, since no feature flags are defined.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`FontSync.h`

FNSFontProfile

Represents a reference to a font profile.

```
typedef struct OpaqueFNSFontProfile * FNSFontProfile;
```

Discussion
The `FNSFontProfile` type is a reference to an opaque structure containing a collection of font references. It defines a set of fonts on the user's system. Although you do not need to use font profiles to iterate, identify, and match fonts on the user's system, they are necessary in building font menus and other font selection human interface elements.

You pass a font profile to FontSync functions that manipulate font profiles. A font reference is passed back by functions that create font profiles. For a description of these functions, see "Working With FontSync Profiles" (page 2426).

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`FontSync.h`

FNSFontReference

Represents a reference to a font reference.

```
typedef struct OpaqueFNSFontReference * FNSFontReference;
```

Discussion
The `FNSFontReference` type is a reference to an opaque structure containing information about a font. Some of the data contained in a font reference includes the QuickDraw font family name, ATSUI-visible font name, type of font, font version, checksums of the data, and information from the font name table.

You pass a font reference to FontSync functions that manipulate font references. A font reference is passed back by functions that create font references. For a description of these functions, see "Working With FontSync References" (page 2426).

Version Notes
Available beginning with FontSync 1.0.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`FontSync.h`

FNSSysInfo

Contains FontSync version and feature information.

```
struct FNSSysInfo {
    FNSObjectVersion iSysInfoVersion;
    FNSFeatureFlags oFeatures;
    FNSObjectVersion oCurRefVersion;
    FNSObjectVersion oMinRefVersion;
    FNSObjectVersion oCurProfileVersion;
    FNSObjectVersion oMinProfileVersion;
    UInt16 oFontSyncVersion;
};
typedef struct FNSSysInfo FNSSysInfo;
```

Fields

`iSysInfoVersion`
: On input, the version of this parameter block structure. In FontSync 1.0, the version number of this structure is 1. Pass the constant `kFNSCurSysInfoVersion`, described in "Version Constants" (page 2462). For more information, see the discussion.

`oFeatures`
: On output, the FontSync features that are available. In FontSync 1.0, no feature flags are defined.

`oCurRefVersion`
: On output, the current font reference format version supported by the FontSync library.

`oMinRefVersion`
: On output, the oldest font reference format version supported by the FontSync library.

`oCurProfileVersion`
: On output, the current font profile format version supported by the FontSync library.

`oMinProfileVersion`
: On output, the oldest font profile format version supported by the FontSync library.

`oFontSyncVersion`
: On output, a binary-coded decimal value indicating the version of FontSync currently running. The high-order 8 bits give the major version, the next four give the minor version, and the last four give the revision. For example, version 1.0 would be encoded as 0x0100.

Discussion

Before calling the function `FNSSysInfoGet` (page 2455), you should fill in the `iSysInfoVersion` field of this structure with the version of this structure. Pass the constant `kFNSCurSysInfoVersion`, described in "Version Constants" (page 2462), to represent the current version. `FNSSysInfoGet` (page 2455) fills in the remaining fields and passes back the structure in the `ioInfo` parameter. The information it provides includes the version of FontSync running in the current context and available features, as well as the current and oldest font reference and profile format versions supported by the FontSync library.

New fields may be added to the end of the structure in future versions of FontSync. FontSync uses the `iSysInfoVersion` field to determine which version of the structure you are using. The value of the current version constant `kFNSCurSysInfoVersion` will change accordingly.

Version Notes

Available beginning with FontSync 1.0. In FontSync 1.0, the value of the `iSysInfoVersion` field is 1. The value of the `oFeatures` field is 0, since no feature flags are defined.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`FontSync.h`

Constants

Matching Options

Represent a mask that you can use to set and determine default match options.

```
typedef UInt32 FNSMatchOptions;
enum {
    kFNSMatchNames = 0x00000001,
    kFNSMatchTechnology = 0x00000002,
    kFNSMatchGlyphs = 0x00000004,
    kFNSMatchEncodings = 0x00000008,
    kFNSMatchQDMetrics = 0x00000010,
    kFNSMatchATSUMetrics = 0x00000020,
    kFNSMatchKerning = 0x00000040,
    kFNSMatchWSLayout = 0x00000080,
    kFNSMatchAATLayout = 0x00000100,
    kFNSMatchPrintEncoding = 0x00000200,
    kFNSMissingDataNoMatch = 0x80000000,
    kFNSMatchAll = 0xFFFFFFFF,
    kFNSMatchDefaults = 0
};
```

Constants

`kFNSMatchNames`

If the bit specified by this mask is set, all significant font names must match. This includes the QuickDraw Text family, ATSUI, unique, full, manufacturer, and version names. Note that the PostScript names are also examined as part of the `kFNSMatchPrintEncoding` option.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchTechnology`

If the bit specified by this mask is set, scaler technologies must match. It is possible to match other parts of the font across different technologies, but this is not supported by FontSync 1.0. As a result, even if this bit is not set, fonts of different technologies will probably not match under any other criteria.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchGlyphs`

If the bit specified by this mask is set, glyph repertoires and outline/bitmap data must match.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchEncodings`
If the bit specified by this mask is set, the `'cmap'` tables must match. If the order of the `'cmap'` tables is different, although the tables are the same, this may be considered a mismatch, since it can cause QuickDraw Text to use a different `'cmap'table`.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchQDMetrics`
If the bit specified by this mask is set, metrics used by QuickDraw Text must match. This includes the effect of `fractEnable` and any metric information in the `'FOND'` resource.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchATSUMetrics`
If the bit specified by this mask is set, metrics used by ATSUI must match. This includes both horizontal and vertical metrics.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchKerning`
If the bit specified by this mask is set, kerning data must match.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchWSLayout`
If the bit specified by this mask is set, layout information given by an `'itl5'` table, whether attached directly to the font or the one provided in the script bundle, must match.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchAATLayout`
If the bit specified by this mask is set, advanced layout information such as that used by ATSUI, must match. This includes such things as ligature and morphing tables. OpenType-style layout information is included in this option.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchPrintEncoding`
If the bit specified by this mask is set, PostScript names and `'FOND'` re-encoding vectors must match. Note that it is an error for a font's internal PostScript name to be different from the one in the `'FOND'`, but FontSync will record both and consider them separately.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMissingDataNoMatch`
If the bit specified by this mask is set, FontSync will report font reference mismatches when both fonts are missing data needed by a selected option. This is useful, since some older fonts may not have all the data needed for matching newer fonts. This makes the mask constant `kFNSMatchAll` specify the most stringent possible match criteria.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSMatchAll`
: If the bit specified by this mask is set, all of the match options must match. In this case, the bit specified by the mask constant `kFNSMissingDataNoMatch` is also set, asserting the most stringent possible match criteria.

  Available in Mac OS X v10.0 and later.

  Declared in `FontSync.h`.

`kFNSMatchDefaults`
: If this constant is specified, the global default match criteria established by the API are used (that is, use all of the options described above in the match). If the user changes the FontSync Control Panel settings, that becomes the new default. This constant basically says to use whatever the user has set.

  Available in Mac OS X v10.0 and later.

  Declared in `FontSync.h`.

Discussion
The `FNSMatchOptions` enumeration defines masks your application can use to set or test match option bits. You can use this mask with the functions `FNSReferenceMatch` (page 2451), `FNSProfileMatchReference` (page 2435), `FNSReferenceMatchFonts` (page 2453), and `FNSReferenceMatchFamilies` (page 2452) to set the match options used during font comparison. If you wish, your application can specify options that are different from the user's preferences via this mask. You can use this mask to test the match option bits produced by the function `FNSMatchDefaultsGet` (page 2427), thereby obtaining the default match options to use in a font comparison. You can also use this mask to test the match option bits produced by the function `FNSReferenceMatch`, thereby determining the match options under consideration that did not match.

At least one of the match options must be set. Having all of these bits clear is equivalent to saying "don't look at anything," which would allow any font to match. Since having all flags clear is nonsensical, the value of the mask constant `kFNSMatchDefaults` is 0. Setting undefined bits does not generate an error and provides backward compatibility.

Version Notes
Available beginning with FontSync 1.0.

Font Profile Constants

Represent the file type and default creator code of a font profile.

```
enum {
    kFNSCreatorDefault = 0,
    kFNSProfileFileType = 'fnsp'
};
```

Constants
`kFNSCreatorDefault`
: Assigns a file creator code instead of using one of your own. Pass this constant in the `iCreator` parameter of the function `FNSProfileCreate` (page 2432).

  Available in Mac OS X v10.0 and later.

  Declared in `FontSync.h`.

`kFNSProfileFileType`
The file type of a profile. It is not used in the API, but is provided for convenience. For example, you can use it to put up a Navigation Services dialog box to select a profile.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

Version Notes
Available beginning with FontSync 1.0. In FontSync 1.0, if you specify the constant `kFNSCreatorDefault` in the `iCreator` parameter of the function `FNSProfileCreate` (page 2432), FontSync assigns the creator code `'fns'`.

Version Constants

Represents version information.

```
typedef UInt32 FNSObjectVersion;
enum {
    kFNSVersionDontCare = 0,
    kFNSCurSysInfoVersion = 1
};
```

Constants
`kFNSVersionDontCare`
Specifies the most recent font reference or font profile version supported by the FontSync library, regardless of version number. In FontSync 1.0, the most recent version for both font references and profiles is version 1. You pass this constant in the `iDesiredVersion` parameter of the functions `FNSSysInfo` (page 2458), `FNSSysInfoGet` (page 2455), and `FNSProfileCreate` (page 2432).

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

`kFNSCurSysInfoVersion`
Identifies the current version of the parameter block structure `FNSSysInfo` (page 2458) returned by the function `FNSSysInfoGet` (page 2455). You pass this constant in the `iSysInfoVersion` field of the system information structure. The version of the structure used by FontSync 1.0 is version 1.

Available in Mac OS X v10.0 and later.

Declared in `FontSync.h`.

Discussion
You can pass the `kFNSVersionDontCare` constant in the `iDesiredVersion` parameter of the functions `FNSReferenceCreate` (page 2441), `FNSReferenceCreateFromFamily` (page 2442), and `FNSProfileCreate` (page 2432), to specify the most recent font reference or font profile version supported by the FontSync library. You can use the `kFNSCurSysInfoVersion` constant in the `iSysInfoVersion` field of the structure `FNSSysInfo` (page 2458) to indicate the current version of the structure.

Result Codes

The most common result codes returned by FontSync are listed below.

Result Code	Value	Description
kFNSInvalidReferenceErr	-29580	Returned by FontSync functions that operate on font references to indicate that the specified font reference is invalid. This can also be returned by the function FNSReferenceUnflatten if the reconstructed reference is bad. Available in Mac OS X v10.0 and later.
kFNSBadReferenceVersionErr	-29581	Returned by FontSync functions that operate on font references to indicate that the reference has an unsupported version number. This may indicate that the reference is valid, but created by a later version of FontSync, or that the reference is truly invalid This is also be returned by font reference creation functions if an unsupported reference version was requested. Available in Mac OS X v10.0 and later.
kFNSInvalidProfileErr	-29582	Returned by FontSync functions that operate on font profiles to indicate that the profile does not have a valid structure. Available in Mac OS X v10.0 and later.
kFNSBadProfileVersionErr	-29583	Returned by FontSync functions that operate on font profiles to indicate that the profile has an unsupported version number. This may indicate that the profile is valid, but created by a later version of FontSync, or that the profile is truly invalid. This is also be returned by font profile creation functions if an unsupported profile version was requested. Available in Mac OS X v10.0 and later.
kFNSDuplicateReferenceErr	-29584	Returned by the function FNSProfileAddReference to indicate that the font reference being added already exists in the profile. Available in Mac OS X v10.0 and later.
kFNSMismatchErr	-29585	Indicates that either a reference did not match, the reference you wish to remove is not in the profile, or that no font names were recorded in the reference. Available in Mac OS X v10.0 and later.
kFNSInsufficientDataErr	-29586	Returned by the functions FNSReferenceMatch, FNSReferenceGetFamilyInfo, FNSReferenceFindName, and FNSReferenceGetIndName to indicate that the mask constant kFNSMissingDataNoMatch has been set and both references being compared are missing the same data. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kFNSBadFlattenedSizeErr	-29587	Returned by the function FNSReferenceUnflatten to indicate that the specified size doesn't match the size recorded in the flattened reference or that the size was not large enough for a flattened reference. Available in Mac OS X v10.0 and later.
kFNSNameNotFoundErr	-29589	Returned by the function FNSReferenceFindName to indicate that there was no name in the font reference that matched the given parameters. Available in Mac OS X v10.0 and later.

Internet Config Reference

Framework: Carbon/Carbon.h

Declared in InternetConfig.h

Overview

Internet Config, a Mac OS 8 and 9 API, supports centralized entry and management of Internet preferences for all of a user's Internet applications. For example, email programs and Web browsers can obtain a user's name, email address, home page, incoming mail server, and similar preferences from one common place that is easily edited by the user via the Internet Config application.

Mac OS X applications should employ Launch Services and System Configuration for managing Internet preferences. In Mac OS X, Internet Config calls through to these newer APIs. Using them directly increases your application's efficiency.

If you use Internet Config in Mac OS X, perhaps to maintain backward compatibility for your application with Mac OS 8 and 9, here are some ways to optimize performance.

- Getting multiple preferences at once.

  Your program may access several Internet Config preferences within a single operation. For example, an FTP program on startup might always get the kICFTPHost, kICFTPProxyHost, kICFTPProxyAccount, kICFTPProxyPassword, and kICFTPProxyUser preferences. Bracket such a sequence of ICGetPref calls with an ICBegin and ICEnd function pair for significantly faster performance.

- Calling Internet Config less often.

  Your application can cache Internet Config preference data and then watch for changes using one of the preference coherency strategies described in the developer's online documentation:

  http://www.quinn.echidna.id.au/Quinn/Config/Prog_Docs.html#PreferenceCoherency

- Using preferences from underlying frameworks.

  Starting with Mac OS X, Internet Config is no longer the final authority on Internet preferences. For example, Launch Services stores URL helper application settings, and System Configuration stores the various proxy settings. When called by an application, Internet Config must call through to the appropriate underlying framework. Call Launch Services and System Configuration directly to improve performance.

The Mac OS 8 and 9 implementation of Internet Config is in the public domain, and Carbon supports all of its commonly-used functions. Functions that are not compatible with preemptive threads are not supported, and Internet Config no longer supports multiple configuration files. This document contains information on replacement functions.

For more information on Internet Config see its developer's web site:

http://www.quinn.echidna.id.au/Quinn/Config

Functions by Task

Starting and Stopping Internet Config

These functions let you create, configure, and destroy connections to Internet Config, denoted by the `ICInstance` type. Although it is usual to create one connection when your program starts and destroy it when it terminates, you can create an arbitrary number of connections at any time.

`ICStart` (page 2509)
: Call this routine to start using Internet Config—typically at application initialization time—passing it your program's creator code.

`ICGetVersion` (page 2502)
: Returns the version of Internet Config installed on the system.

`ICStop` (page 2509)
: Call this when your application is done using Internet Config, passing it the instance you got from `ICStart`.

Getting Information About an Instance

`ICGetConfigName` (page 2496)
: Returns a displayable string that represents the instance's current configuration.

`ICGetSeed` (page 2501)
: Returns the seed for the current preferences set.

`ICGetPerm` (page 2500)
: Returns the current permissions for this instance, i.e. the permission value you used when you called `ICBegin`, or returns `ioNoPerm` if you haven't call `ICBegin`.

Preparing to Read and Write Preferences

These routines are not always required because the commonly used reading and writing calls perform this operation automatically. However, even in that case, these routines are useful if you are making repeated calls because they allow those calls to work faster.

`ICBegin` (page 2473)
: Prepares Internet Config to read (and, optionally, to write) preferences.

`ICEnd` (page 2494)
: Tells Internet Config that you have finished accessing preference information.

Reading and Writing Preferences

`ICGetPref` (page 2500)
: Given a preference's key, gets the preference's data and places it into a buffer that you supply.

`ICSetPref` (page 2507)
Sets a preference given its key, attributes, and a buffer containing the preference data.

`ICFindPrefHandle` (page 2495)
Gets a preference's data, like `ICGetPref`, but returns the data in a handle.

`ICSetPrefHandle` (page 2508)
Sets a preference, like `ICSetPref`, but takes its input as a handle.

`ICGetDefaultPref` (page 2498)
Gets the default value for the preference associated with the specified key.

`ICDeletePref` (page 2493)
Deletes the preference for the specified key.

Enumerating All Preferences

You must call `ICBegin` before calling any of these routines.

`ICCountPref` (page 2487)
Returns the total number of preferences available.

`ICGetIndPref` (page 2498)
Returns the key associated with the preference at the specified index.

Accessing the User Interface

We recommend that you do not provide a user interface for editing Internet Config preferences from within your application. In Mac OS X, simply provide a way for the user to open the Internet pane of System Preferences. The `ICEditPreferences` function provides support for this.

`ICEditPreferences` (page 2494)
Opens the Internet pane of System Preferences.

URL Functions

`ICParseURL` (page 2505)
Parses a URL out of the specified text and returns it to the calling program.

`ICLaunchURL` (page 2502)
Parses a URL out of the specified text and feeds it to the appropriate helper application.

`ICCreateGURLEvent` (page 2489)
Creates a GURL Apple event, targetted at the application with the specified creator code

`ICSendGURLEvent` (page 2506)
Sends a specified Apple Event to the target application.

Mapping-Database High-Level Functions

Internet Config's high-level functions are suitable for applications that want to easily look up a file type and creator based on an extension, or vice versa. These functions are significantly slower than their lower level counterparts, especially if you call them repeatedly.

`ICMapFilename` (page 2504)
Given a filename, returns the most appropriate `ICMapEntry` based on the filename extension.

`ICMapTypeCreator` (page 2504)
Given a file type and creator (and optionally a filename), returns the most appropriate `ICMapEntry`.

Mapping-Database Mid-Level Functions

These functions are useful if you are doing multiple searches because they avoid the overhead of accessing the mappings database each time.

`ICMapEntriesFilename` (page 2503)
Given a filename, returns the most appropriate `ICMapEntry` based on the filename extension—without the overhead of a mappings database access.

`ICMapEntriesTypeCreator` (page 2503)
Given a file type and creator (and optionally a filename), returns the most appropriate `ICMapEntry`—without the overhead of a mappings database access.

Mapping-Database Low-Level Functions

Internet Config's low-level functions give you access to the primitive operations used to implement the other mapping functions.

`ICCountMapEntries` (page 2487)
Counts the number of entries in the mappings database preference provided.

`ICGetIndMapEntry` (page 2498)
Returns the mappings database entry for a specified index .

`ICGetMapEntry` (page 2499)
Returns a mappings database entry based on its position in the Mapping preference.

`ICSetMapEntry` (page 2507)
Sets an entry in a specified Mapping preference.

`ICDeleteMapEntry` (page 2493)
Deletes an entry in the specified Mapping preference.

`ICAddMapEntry` (page 2472)
Adds an entry to the mappings database.

Profile Functions

`ICGetCurrentProfile` (page 2497)
Returns the profile ID of the current profile.

`ICSetCurrentProfile` (page 2506)
Sets the current profile to a specified profile ID.

`ICCountProfiles` (page 2488)
Returns the number of available profiles.

`ICGetIndProfile` (page 2499)
Returns the ID of the profile at the specified index.

`ICGetProfileName` (page 2501)
Returns the name of the profile that has the specified ID.

`ICSetProfileName` (page 2508)
Sets the name of the profile that has the specified ID.

`ICAddProfile` (page 2473)
Creates a new profile and returns its ID.

`ICDeleteProfile` (page 2494)
Deletes the profile that has the specified ID.

Deprecated functions

`ICGetPrefHandle` (page 2501)
Deprecated. While this function works in Mac OS X, you should use ICFindPrefHandle instead.

Unsupported Functions

`ICCSetPref` (page 2491)

`ICCAddMapEntry` (page 2473)

`ICCAddProfile` (page 2474)

`ICCBegin` (page 2474)

`ICCChooseConfig` (page 2474)

`ICCChooseNewConfig` (page 2474)

`ICCCountMapEntries` (page 2475)

`ICCCountPref` (page 2475)

`ICCCountProfiles` (page 2475)

`ICCCreateGURLEvent` (page 2476)

`ICCDefaultFileName` (page 2476)

`ICCDeleteMapEntry` (page 2476)

`ICCDeletePref` (page 2477)

Functions

ICAddMapEntry

Adds an entry to the mappings database.

```
OSStatus ICAddMapEntry (
   ICInstance inst,
   Handle entries,
   const ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICAddProfile

Creates a new profile and returns its ID.

```
OSStatus ICAddProfile (
   ICInstance inst,
   ICProfileID prototypeID,
   ICProfileID *newID
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICBegin

Prepares Internet Config to read (and, optionally, to write) preferences.

```
OSStatus ICBegin (
   ICInstance inst,
   ICPerm perm
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICCAddMapEntry

Unsupported

```
OSStatus ICCAddMapEntry (
    ComponentInstance inst,
    Handle entries,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICAddMapEntry` (page 2472) instead.

Declared In
`InternetConfig.h`

ICCAddProfile

Unsupported

```
OSStatus ICCAddProfile (
    ComponentInstance inst,
    ICProfileID prototypeID,
    ICProfileID *newID
);
```

Carbon Porting Notes

Use `ICAddProfile` (page 2473) instead.

Declared In

`InternetConfig.h`

ICCBegin

Unsupported

```
OSStatus ICCBegin (
    ComponentInstance inst,
    ICPerm perm
);
```

Carbon Porting Notes

Use `ICBegin` (page 2473) instead.

Declared In

`InternetConfig.h`

ICCChooseConfig

Unsupported

```
OSStatus ICCChooseConfig (
    ComponentInstance inst
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In

`InternetConfig.h`

ICCChooseNewConfig

Unsupported

```
OSStatus ICCChooseNewConfig (
    ComponentInstance inst
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In

`InternetConfig.h`

ICCCountMapEntries

Unsupported

```
OSStatus ICCCountMapEntries (
    ComponentInstance inst,
    Handle entries,
    SInt32 *count
);
```

Carbon Porting Notes

Use `ICCountMapEntries` (page 2487) instead.

Declared In

`InternetConfig.h`

ICCCountPref

Unsupported

```
OSStatus ICCCountPref (
    ComponentInstance inst,
    SInt32 *count
);
```

Carbon Porting Notes

Use `ICCountPref` (page 2487) instead.

Declared In

`InternetConfig.h`

ICCCountProfiles

Unsupported

```
OSStatus ICCCountProfiles (
    ComponentInstance inst,
    SInt32 *count
);
```

Carbon Porting Notes

Use `ICCountProfiles` (page 2488) instead.

Declared In
InternetConfig.h

ICCCreateGURLEvent
Unsupported

```
OSStatus ICCCreateGURLEvent (
    ComponentInstance inst,
    OSType helperCreator,
    Handle urlH,
    AppleEvent *theEvent
);
```

Carbon Porting Notes

Use ICCreateGURLEvent (page 2489) instead.

Declared In
InternetConfig.h

ICCDefaultFileName
Unsupported

```
OSStatus ICCDefaultFileName (
    ComponentInstance inst,
    Str63 name
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In
InternetConfig.h

ICCDeleteMapEntry
Unsupported

```
OSStatus ICCDeleteMapEntry (
    ComponentInstance inst,
    Handle entries,
    SInt32 pos
);
```

Carbon Porting Notes

Use ICDeleteMapEntry (page 2493) instead.

Declared In
InternetConfig.h

ICCDeletePref

Unsupported

```
OSStatus ICCDeletePref (
    ComponentInstance inst,
    ConstStr255Param key
);
```

Carbon Porting Notes

Use `ICDeletePref` (page 2493) instead.

Declared In

`InternetConfig.h`

ICCDeleteProfile

Unsupported

```
OSStatus ICCDeleteProfile (
    ComponentInstance inst,
    ICProfileID thisID
);
```

Carbon Porting Notes

Use `ICDeleteProfile` (page 2494) instead.

Declared In

`InternetConfig.h`

ICCEditPreferences

Unsupported

```
OSStatus ICCEditPreferences (
    ComponentInstance inst,
    ConstStr255Param key
);
```

Carbon Porting Notes

Use `ICEditPreferences` (page 2494) instead.

Declared In

`InternetConfig.h`

ICCEnd

Unsupported

```
OSStatus ICCEnd (
    ComponentInstance inst
);
```

Carbon Porting Notes

Use `ICEnd` (page 2494) instead.

Declared In

`InternetConfig.h`

ICCFindConfigFile

Unsupported

```
OSStatus ICCFindConfigFile (
    ComponentInstance inst,
    SInt16 count,
    ICDirSpecArrayPtr folders
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In

`InternetConfig.h`

ICCFindPrefHandle

Unsupported

```
OSStatus ICCFindPrefHandle (
    ComponentInstance inst,
    ConstStr255Param key,
    ICAttr *attr,
    Handle prefh
);
```

Carbon Porting Notes

Use `ICFindPrefHandle` (page 2495) instead.

Declared In

`InternetConfig.h`

ICCFindUserConfigFile

Unsupported

```
OSStatus ICCFindUserConfigFile (
    ComponentInstance inst,
    ICDirSpec *where
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In

`InternetConfig.h`

ICCGeneralFindConfigFile

Unsupported

```
OSStatus ICCGeneralFindConfigFile (
    ComponentInstance inst,
    Boolean searchPrefs,
    Boolean canCreate,
    SInt16 count,
    ICDirSpecArrayPtr folders
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In

`InternetConfig.h`

ICCGetComponentInstance

Unsupported

```
OSStatus ICCGetComponentInstance (
    ComponentInstance inst,
    ComponentInstance *componentInst
);
```

Carbon Porting Notes

Because Internet Config is not component-based under Mac OS X, use `ICGetVersion` (page 2502) instead.

Declared In

`InternetConfig.h`

ICCGetConfigName

Unsupported

```
OSStatus ICCGetConfigName (
    ComponentInstance inst,
    Boolean longname,
    Str255 name
);
```

Carbon Porting Notes

Use `ICGetConfigName` (page 2496) instead.

Declared In

`InternetConfig.h`

ICCGetConfigReference

Unsupported

```
OSStatus ICCGetConfigReference (
    ComponentInstance inst,
    ICConfigRefHandle ref
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In

`InternetConfig.h`

ICCGetCurrentProfile

Unsupported

```
OSStatus ICCGetCurrentProfile (
    ComponentInstance inst,
    ICProfileID *currentID
);
```

Carbon Porting Notes

Use `ICGetCurrentProfile` (page 2497) instead.

Declared In

`InternetConfig.h`

ICCGetDefaultPref

Unsupported

```
OSStatus ICCGetDefaultPref (
    ComponentInstance inst,
    ConstStr255Param key,
    Handle prefH
);
```

Carbon Porting Notes

Use `ICGetDefaultPref` (page 2498) instead.

Declared In

`InternetConfig.h`

ICCGetIndMapEntry

Unsupported

```
OSStatus ICCGetIndMapEntry (
    ComponentInstance inst,
    Handle entries,
    SInt32 index,
    SInt32 *pos,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICGetIndMapEntry` (page 2498) instead.

Declared In

`InternetConfig.h`

ICCGetIndPref

Unsupported

```
OSStatus ICCGetIndPref (
    ComponentInstance inst,
    SInt32 index,
    Str255 key
);
```

Carbon Porting Notes

Use `ICGetIndPref` (page 2498) instead.

Declared In

`InternetConfig.h`

ICCGetIndProfile

Unsupported

```
OSStatus ICCGetIndProfile (
    ComponentInstance inst,
    SInt32 index,
    ICProfileID *thisID
);
```

Carbon Porting Notes

Use `ICGetIndProfile` (page 2499) instead.

Declared In

`InternetConfig.h`

ICCGetMapEntry

Unsupported

```
OSStatus ICCGetMapEntry (
    ComponentInstance inst,
    Handle entries,
    SInt32 pos,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICGetMapEntry` (page 2499) instead.

Declared In

`InternetConfig.h`

ICCGetMappingInterruptSafe

Unsupported

```
OSStatus ICCGetMappingInterruptSafe (
    ComponentInstance inst,
    Ptr *mappingPref,
    SInt32 *mappingPrefSize
);
```

Carbon Porting Notes

`ICCGetMappingInterruptSafe` is not compatible with preemptive threads. If your application relies on this API, contact Apple Developer Technical Support.

Declared In

`InternetConfig.h`

ICCGetPerm

Unsupported

```
OSStatus ICCGetPerm (
    ComponentInstance inst,
    ICPerm *perm
);
```

Carbon Porting Notes

Use `ICGetPerm` (page 2500) instead.

Declared In

`InternetConfig.h`

ICCGetPref

Unsupported

```
OSStatus ICCGetPref (
    ComponentInstance inst,
    ConstStr255Param key,
    ICAttr *attr,
    Ptr buf,
    SInt32 *size
);
```

Carbon Porting Notes

Use `ICGetPref` (page 2500) instead.

Declared In

`InternetConfig.h`

ICCGetPrefHandle

Unsupported

```
OSStatus ICCGetPrefHandle (
    ComponentInstance inst,
    ConstStr255Param key,
    ICAttr *attr,
    Handle *prefh
);
```

Carbon Porting Notes

Use `ICFindPrefHandle` (page 2495) instead.

Declared In

`InternetConfig.h`

ICCGetProfileName

Unsupported

```
OSStatus ICCGetProfileName (
    ComponentInstance inst,
    ICProfileID thisID,
    Str255 name
);
```

Carbon Porting Notes

Use `ICGetProfileName` (page 2501) instead.

Declared In

`InternetConfig.h`

ICCGetSeed

Unsupported

```
OSStatus ICCGetSeed (
    ComponentInstance inst,
    SInt32 *seed
);
```

Carbon Porting Notes

Use `ICGetSeed` (page 2501) instead.

Declared In

`InternetConfig.h`

ICCGetSeedInterruptSafe

Unsupported

```
OSStatus ICCGetSeedInterruptSafe (
    ComponentInstance inst,
    SInt32 *seed
);
```

Carbon Porting Notes

`ICCGetSeedInterruptSafe` is not compatible with preemptive threads. If your application relies on this API, contact Apple Developer Technical Support.

Declared In

`InternetConfig.h`

ICCGetVersion

Unsupported

```
OSStatus ICCGetVersion (
    ComponentInstance inst,
    SInt32 whichVersion,
    UInt32 *version
);
```

Carbon Porting Notes

Use `ICGetVersion` (page 2502) instead.

Declared In

`InternetConfig.h`

ICChooseConfig

Unsupported

```
OSStatus ICChooseConfig (
    ICInstance inst
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In

`InternetConfig.h`

ICChooseNewConfig

Unsupported

```
OSStatus ICChooseNewConfig (
    ICInstance inst
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In

`InternetConfig.h`

ICCLaunchURL

Unsupported

```
OSStatus ICCLaunchURL (
    ComponentInstance inst,
    ConstStr255Param hint,
    Ptr data,
    SInt32 len,
    SInt32 *selStart,
    SInt32 *selEnd
);
```

Carbon Porting Notes

Use `ICLaunchURL` (page 2502) instead.

Declared In

`InternetConfig.h`

ICCMapEntriesFilename

Unsupported

```
OSStatus ICCMapEntriesFilename (
    ComponentInstance inst,
    Handle entries,
    ConstStr255Param filename,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICMapEntriesFilename` (page 2503) instead.

Declared In

`InternetConfig.h`

ICCMapEntriesTypeCreator

Unsupported

```
OSStatus ICCMapEntriesTypeCreator (
    ComponentInstance inst,
    Handle entries,
    OSType fType,
    OSType fCreator,
    ConstStr255Param filename,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICMapEntriesTypeCreator` (page 2503) instead.

Declared In

`InternetConfig.h`

ICCMapFilename

Unsupported

```
OSStatus ICCMapFilename (
    ComponentInstance inst,
    ConstStr255Param filename,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICMapFilename` (page 2504) instead.

Declared In

`InternetConfig.h`

ICCMapTypeCreator

Unsupported

```
OSStatus ICCMapTypeCreator (
    ComponentInstance inst,
    OSType fType,
    OSType fCreator,
    ConstStr255Param filename,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICMapTypeCreator` (page 2504) instead.

Declared In

`InternetConfig.h`

ICCountMapEntries

Counts the number of entries in the mappings database preference provided.

```
OSStatus ICCountMapEntries (
   ICInstance inst,
   Handle entries,
   long *count
);
```

Return Value

A result code. See "Result Codes" (page 2538).

Availability

Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`InternetConfig.h`

ICCountPref

Returns the total number of preferences available.

```
OSStatus ICCountPref (
   ICInstance inst,
   long *count
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICCountProfiles

Returns the number of available profiles.

```
OSStatus ICCountProfiles (
   ICInstance inst,
   long *count
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICCParseURL

Unsupported

```
OSStatus ICCParseURL (
    ComponentInstance inst,
    ConstStr255Param hint,
    Ptr data,
    SInt32 len,
    SInt32 *selStart,
    SInt32 *selEnd,
    Handle url
);
```

Carbon Porting Notes

Use `ICParseURL` (page 2505) instead.

Declared In
`InternetConfig.h`

ICCreateGURLEvent

Creates a GURL Apple event, targetted at the application with the specified creator code

```
OSStatus ICCreateGURLEvent (
   ICInstance inst,
   OSType helperCreator,
   Handle urlH,
   AppleEvent *theEvent
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICCRefreshCaches

Unsupported

```
OSStatus ICCRefreshCaches (
    ComponentInstance inst
);
```

Carbon Porting Notes

This function is obsolete, because there is no supported way to modify the Internet Config database without going through the Internet Config API. There is no replacement function.

Declared In
`InternetConfig.h`

ICCRequiresInterruptSafe

Unsupported

```
OSStatus ICCRequiresInterruptSafe (
    ComponentInstance inst
);
```

Carbon Porting Notes

This function is not compatible with preemptive threads. There is no replacement.

Declared In
`InternetConfig.h`

ICCSendGURLEvent

Unsupported

```
OSStatus ICCSendGURLEvent (
    ComponentInstance inst,
    AppleEvent *theEvent
);
```

Carbon Porting Notes

Use `ICSendGURLEvent` (page 2506) instead.

Declared In

`InternetConfig.h`

ICCSetConfigReference

Unsupported

```
OSStatus ICCSetConfigReference (
    ComponentInstance inst,
    ICConfigRefHandle ref,
    SInt32 flags
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In

`InternetConfig.h`

ICCSetCurrentProfile

Unsupported

```
OSStatus ICCSetCurrentProfile (
    ComponentInstance inst,
    ICProfileID newID
);
```

Carbon Porting Notes

Use `ICSetCurrentProfile` (page 2506) instead.

Declared In

`InternetConfig.h`

ICCSetMapEntry

Unsupported

```
OSStatus ICCSetMapEntry (
    ComponentInstance inst,
    Handle entries,
    SInt32 pos,
    ICMapEntry *entry
);
```

Carbon Porting Notes

Use `ICSetMapEntry` (page 2507)`ICSetMapEntry` instead.

Declared In

`InternetConfig.h`

ICCSetPref

Unsupported

```
OSStatus ICCSetPref (
    ComponentInstance inst,
    ConstStr255Param key,
    ICAttr attr,
    Ptr buf,
    SInt32 size
);
```

Carbon Porting Notes

Use `ICSetPref` (page 2507) instead.

Declared In

`InternetConfig.h`

ICCSetPrefHandle

Unsupported

```
OSStatus ICCSetPrefHandle (
    ComponentInstance inst,
    ConstStr255Param key,
    ICAttr attr,
    Handle prefh
);
```

Carbon Porting Notes

Use `ICSetPrefHandle` (page 2508) instead.

Declared In

`InternetConfig.h`

ICCSetProfileName

Unsupported

```
OSStatus ICCSetProfileName (
    ComponentInstance inst,
    ICProfileID thisID,
    ConstStr255Param name
);
```

Carbon Porting Notes

Use `ICSetProfileName` (page 2508) instead.

Declared In

`InternetConfig.h`

ICCSpecifyConfigFile

Unsupported

```
OSStatus ICCSpecifyConfigFile (
    ComponentInstance inst,
    FSSpec *config
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In

`InternetConfig.h`

ICCStart

Unsupported

```
OSStatus ICCStart (
    ComponentInstance *inst,
    OSType creator
);
```

Carbon Porting Notes

Use `ICStart` (page 2509) instead.

Declared In

`InternetConfig.h`

ICCStop

Unsupported

```
OSStatus ICCStop (
    ComponentInstance inst
);
```

Carbon Porting Notes

Use `ICStop` (page 2509) instead.

Declared In
InternetConfig.h

ICDefaultFileName

Unsupported

```
OSStatus ICDefaultFileName (
    ICInstance inst,
    Str63 name
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In
InternetConfig.h

ICDeleteMapEntry

Deletes an entry in the specified Mapping preference.

```
OSStatus ICDeleteMapEntry (
   ICInstance inst,
   Handle entries,
   long pos
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
InternetConfig.h

ICDeletePref

Deletes the preference for the specified key.

```
OSStatus ICDeletePref (
   ICInstance inst,
   ConstStr255Param key
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICDeleteProfile

Deletes the profile that has the specified ID.

```
OSStatus ICDeleteProfile (
   ICInstance inst,
   ICProfileID thisID
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICEditPreferences

Opens the Internet pane of System Preferences.

```
OSStatus ICEditPreferences (
   ICInstance inst,
   ConstStr255Param key
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Discussion
This function launches the Internet preference control appropriate to the operating system version. In Mac OS X, it opens the Internet pane of System Preferences. In Mac OS 8 and 9 wiith Carbon, it launches the Internet control panel. On earlier systems, it launches Internet Config.

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICEnd

Tells Internet Config that you have finished accessing preference information.

```
OSStatus ICEnd (
   ICInstance inst
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICFindConfigFile

Unsupported

```
OSStatus ICFindConfigFile (
    ICInstance inst,
    SInt16 count,
    ICDirSpecArrayPtr folders
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In
`InternetConfig.h`

ICFindPrefHandle

Gets a preference's data, like `ICGetPref`, but returns the data in a handle.

```
OSStatus ICFindPrefHandle (
   ICInstance inst,
   ConstStr255Param key,
   ICAttr *attr,
   Handle prefh
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICFindUserConfigFile

Unsupported

```
OSStatus ICFindUserConfigFile (
    ICInstance inst,
    ICDirSpec *where
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In

`InternetConfig.h`

ICGeneralFindConfigFile

Unsupported

```
OSStatus ICGeneralFindConfigFile (
    ICInstance inst,
    Boolean searchPrefs,
    Boolean canCreate,
    SInt16 count,
    ICDirSpecArrayPtr folders
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In

`InternetConfig.h`

ICGetComponentInstance

Unsupported

```
OSStatus ICGetComponentInstance (
    ICInstance inst,
    ComponentInstance *componentInst
);
```

Carbon Porting Notes

Because Internet Config is not component-based under X, use `ICGetVersion` instead of `ICGetComponentInstance`.

Declared In

`InternetConfig.h`

ICGetConfigName

Returns a displayable string that represents the instance's current configuration.

```
OSStatus ICGetConfigName (
   ICInstance inst,
   Boolean longname,
   Str255 name
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetConfigReference

Unsupported

```
OSStatus ICGetConfigReference (
    ICInstance inst,
    ICConfigRefHandle ref
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In
`InternetConfig.h`

ICGetCurrentProfile

Returns the profile ID of the current profile.

```
OSStatus ICGetCurrentProfile (
   ICInstance inst,
   ICProfileID *currentID
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetDefaultPref

Gets the default value for the preference associated with the specified key.

```
OSStatus ICGetDefaultPref (
   ICInstance inst,
   ConstStr255Param key,
   Handle prefH
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetIndMapEntry

Returns the mappings database entry for a specified index .

```
OSStatus ICGetIndMapEntry (
   ICInstance inst,
   Handle entries,
   long index,
   long *pos,
   ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetIndPref

Returns the key associated with the preference at the specified index.

```
OSStatus ICGetIndPref (
   ICInstance inst,
   long index,
   Str255 key
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetIndProfile

Returns the ID of the profile at the specified index.

```
OSStatus ICGetIndProfile (
   ICInstance inst,
   long index,
   ICProfileID *thisID
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetMapEntry

Returns a mappings database entry based on its position in the Mapping preference.

```
OSStatus ICGetMapEntry (
   ICInstance inst,
   Handle entries,
   long pos,
   ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetMappingInterruptSafe

Unsupported

```
OSStatus ICGetMappingInterruptSafe (
    ICInstance inst,
    Ptr *mappingPref,
    SInt32 *mappingPrefSize
);
```

Carbon Porting Notes

Because `ICGetMappingInterruptSafe` is not compatible with preemptive threads, it has been removed from Carbon.

Declared In
`InternetConfig.h`

ICGetPerm

Returns the current permissions for this instance, i.e. the permission value you used when you called `ICBegin`, or returns `ioNoPerm` if you haven't call `ICBegin`.

```
OSStatus ICGetPerm (
   ICInstance inst,
   ICPerm *perm
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetPref

Given a preference's key, gets the preference's data and places it into a buffer that you supply.

```
OSStatus ICGetPref (
   ICInstance inst,
   ConstStr255Param key,
   ICAttr *attr,
   void *buf,
   long *size
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetPrefHandle

Deprecated. While this function works in Mac OS X, you should use ICFindPrefHandle instead.

```
OSStatus ICGetPrefHandle (
   ICInstance inst,
   ConstStr255Param key,
   ICAttr *attr,
   Handle *prefh
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetProfileName

Returns the name of the profile that has the specified ID.

```
OSStatus ICGetProfileName (
   ICInstance inst,
   ICProfileID thisID,
   Str255 name
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetSeed

Returns the seed for the current preferences set.

```
OSStatus ICGetSeed (
   ICInstance inst,
   long *seed
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICGetSeedInterruptSafe

Unsupported

```
OSStatus ICGetSeedInterruptSafe (
    ICInstance inst,
    SInt32 *seed
);
```

Carbon Porting Notes

Because `ICGetSeedInterruptSafe` is not compatible with preemptive threads, it has been removed from Carbon.

Declared In
`InternetConfig.h`

ICGetVersion

Returns the version of Internet Config installed on the system.

```
OSStatus ICGetVersion (
   ICInstance inst,
   long whichVersion,
   UInt32 *version
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Discussion
For the range of versions, refer to "Version Constants" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICLaunchURL

Parses a URL out of the specified text and feeds it to the appropriate helper application.

```
OSStatus ICLaunchURL (
   ICInstance inst,
   ConstStr255Param hint,
   const void *data,
   long len,
   long *selStart,
   long *selEnd
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICMapEntriesFilename

Given a filename, returns the most appropriate `ICMapEntry` based on the filename extension—without the overhead of a mappings database access.

```
OSStatus ICMapEntriesFilename (
   ICInstance inst,
   Handle entries,
   ConstStr255Param filename,
   ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICMapEntriesTypeCreator

Given a file type and creator (and optionally a filename), returns the most appropriate `ICMapEntry`—without the overhead of a mappings database access.

```
OSStatus ICMapEntriesTypeCreator (
   ICInstance inst,
   Handle entries,
   OSType fType,
   OSType fCreator,
   ConstStr255Param filename,
   ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICMapFilename

Given a filename, returns the most appropriate `ICMapEntry` based on the filename extension.

```
OSStatus ICMapFilename (
   ICInstance inst,
   ConstStr255Param filename,
   ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICMapTypeCreator

Given a file type and creator (and optionally a filename), returns the most appropriate `ICMapEntry`.

```
OSStatus ICMapTypeCreator (
   ICInstance inst,
   OSType fType,
   OSType fCreator,
   ConstStr255Param filename,
   ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICParseURL

Parses a URL out of the specified text and returns it to the calling program.

```
OSStatus ICParseURL (
   ICInstance inst,
   ConstStr255Param hint,
   const void *data,
   long len,
   long *selStart,
   long *selEnd,
   Handle url
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICRefreshCaches

Unsupported

```
OSStatus ICRefreshCaches (
    ICInstance inst
);
```

Carbon Porting Notes

This function `ICRefreshCaches` is obsolete. There is no supported way to modify the Internet Config database without going through the Internet Config API. There is no replacement function.

Declared In
`InternetConfig.h`

ICRequiresInterruptSafe

Unsupported

```
OSStatus ICRequiresInterruptSafe (
    ICInstance inst
);
```

Carbon Porting Notes

Because `ICRequiresInterruptSafe` is not compatible with preemptive threads, it has been removed from Carbon.

Declared In
`InternetConfig.h`

ICSendGURLEvent

Sends a specified Apple Event to the target application.

```
OSStatus ICSendGURLEvent (
   ICInstance inst,
   AppleEvent *theEvent
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICSetConfigReference

Unsupported

```
OSStatus ICSetConfigReference (
    ICInstance inst,
    ICConfigRefHandle ref,
    SInt32 flags
);
```

Carbon Porting Notes

Because Internet Config no longer supports multiple configuration files, this function is obsolete, and there is no replacement function.

Declared In
`InternetConfig.h`

ICSetCurrentProfile

Sets the current profile to a specified profile ID.

```
OSStatus ICSetCurrentProfile (
   ICInstance inst,
   ICProfileID newID
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICSetMapEntry

Sets an entry in a specified Mapping preference.

```
OSStatus ICSetMapEntry (
   ICInstance inst,
   Handle entries,
   long pos,
   const ICMapEntry *entry
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICSetPref

Sets a preference given its key, attributes, and a buffer containing the preference data.

```
OSStatus ICSetPref (
   ICInstance inst,
   ConstStr255Param key,
   ICAttr attr,
   const void *buf,
   long size
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICSetPrefHandle

Sets a preference, like `ICSetPref`, but takes its input as a handle.

```
OSStatus ICSetPrefHandle (
   ICInstance inst,
   ConstStr255Param key,
   ICAttr attr,
   Handle prefh
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICSetProfileName

Sets the name of the profile that has the specified ID.

```
OSStatus ICSetProfileName (
   ICInstance inst,
   ICProfileID thisID,
   ConstStr255Param name
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICSpecifyConfigFile

Unsupported

```
OSStatus ICSpecifyConfigFile (
    ICInstance inst,
    FSSpec *config
);
```

Carbon Porting Notes

Functions related to finding different IC database files are obsolete. You can simply remove these calls from your code—there are no replacement functions.

Declared In
`InternetConfig.h`

ICStart

Call this routine to start using Internet Config—typically at application initialization time—passing it your program's creator code.

```
OSStatus ICStart (
   ICInstance *inst,
   OSType signature
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

ICStop

Call this when your application is done using Internet Config, passing it the instance you got from `ICStart`.

```
OSStatus ICStop (
   ICInstance inst
);
```

Return Value
A result code. See "Result Codes" (page 2538).

Availability
Available in CarbonLib 1.0.2 and later when Internet Config 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`InternetConfig.h`

Data Types

ICAppSpec

A fixed length record used to specify an application.

```
struct ICAppSpec {
    OSType fCreator;
    Str63 name;
};
typedef struct ICAppSpec ICAppSpec;
typedef ICAppSpec * ICAppSpecPtr;
typedef ICAppSpecPtr * ICAppSpecHandle;
```

Discussion
The program using this specification is expected to look up the location of the application in the desktop database. The name is provided to display to the user, and should not affect the search.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICAppSpecList

Represents a list of application specifications.

```
struct ICAppSpecList {
    short numberOfItems;
    ICAppSpec appSpecs[1];
};
typedef struct ICAppSpecList ICAppSpecList;
typedef ICAppSpecList * ICAppSpecListPtr;
typedef ICAppSpecListPtr * ICAppSpecListHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICCharTable

Specifies a mapping from MacRoman encoding (kTextEncodingMacRoman) to Mac Net ASCII (kTextEncodingMacRomanLatin1) and vice versa.

```
struct ICCharTable {
    unsigned char netToMac[256];
    unsigned char macToNet[256];
};
typedef struct ICCharTable ICCharTable;
typedef ICCharTable * ICCharTablePtr;
typedef ICCharTablePtr * ICCharTableHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICConfigRef

Stores a permanent reference to an Internet Config configuration.

```
struct ICConfigRef {
    OSType manufacturer;
};
typedef struct ICConfigRef ICConfigRef;
typedef ICConfigRef * ICConfigRefPtr;
typedef ICConfigRefPtr * ICConfigRefHandle;
```

Discussion
The `ICConfigRef` type varies in length and only the first four bytes, representing the manufacturer field, has a public meaning. For more information see the IC Internals Documentation.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`InternetConfig.h`

ICDirSpec

Holds the `vRefNum` and `dirID` of a directory.

```
struct ICDirSpec {
    short vRefNum;
    long dirID;
};
typedef struct ICDirSpec ICDirSpec;
typedef ICDirSpec ICDirSpecArray[4];
typedef ICDirSpecArray * ICDirSpecArrayPtr;
```

Discussion
An array of `ICDirSpec` records is supplied to the pre-Carbon-only function `ICFindConfigFile` to specify the search path. This array is defined to contain just 4 elements, but is in fact arbitrarily extensible.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`InternetConfig.h`

ICError

Used for all error results from Internet Config.

```
typedef long ICError;
```

Discussion
ICError uses a long integer because Internet Config makes calls to Component Manager, which uses long integers for error codes.

Special Considerations

This data type is available only if you define `OLDROUTINENAMES`.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`InternetConfig.h`

ICFileInfo

Defines the format of a key.

```
struct ICFileInfo {
    OSType fType;
    OSType fCreator;
    Str63 name;
};
typedef struct ICFileInfo ICFileInfo;
typedef ICFileInfo * ICFileInfoPtr;
typedef ICFileInfoPtr * ICFileInfoHandle;
```

Special Considerations

This data type defines the format of a key. That key data type has previouisly been removed from the header file. `ICFileInfo` is deprecated and will also be removed, but for the moment, it is available if you define `OLDROUTINENAMES`.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`InternetConfig.h`

ICFileSpec

A variable length data structure used to specify a file or folder.

```
struct ICFileSpec {
    Str31 volName;
    long volCreationDate;
    FSSpec fss;
    AliasRecord alias;
};
typedef struct ICFileSpec ICFileSpec;
typedef ICFileSpec * ICFileSpecPtr;
typedef ICFileSpecPtr * ICFileSpecHandle;
```

Discussion
`ICFileSpec` contains both an alias and a 'poor man's alias.' All modern applications can use the real alias and ignore the poor man's alias. The latter was included so that programs running under System 6 could specify file positions using `vol_name` and `vol_creation_date` for the volume, `fss.parID` for the directory on the volume, and `fss.name` for the file in the directory.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICFontRecord

A fixed length record used to specify a font, size and face.

```
struct ICFontRecord {
    short size;
    Style face;
    char pad;
    Str255 font;
};
typedef struct ICFontRecord ICFontRecord;
typedef ICFontRecord * ICFontRecordPtr;
typedef ICFontRecordPtr * ICFontRecordHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICInstance

An opaque type used to hold a reference to a session with Internet Config.

```
typedef struct OpaqueICInstance * ICInstance;
```

Discussion
Applications can create IC instances by calling `ICStart`, use them with any of the API routines, and destroy them by calling `ICStop`.

An `ICInstance` is a pointer, so you can use the system `nil` value to denote an invalid instance. Do not pass an `ICInstance` between processes. Also, do not pass an `ICInstance` between instruction set architectures, i.e. from PowerPC to 68K or vice versa.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICMapEntry

Specifies an Internet Config map entry.

```
struct ICMapEntry {
    short totalLength;
    ICFixedLength fixedLength;
    short version;
    OSType fileType;
    OSType fileCreator;
    OSType postCreator;
    ICMapEntryFlags flags;
    Str255 extension;
    Str255 creatorAppName;
    Str255 postAppName;
    Str255 MIMEType;
    Str255 entryName;
};
typedef struct ICMapEntry ICMapEntry;
typedef ICMapEntry * ICMapEntryPtr;
typedef ICMapEntryPtr * ICMapEntryHandle;
```

Fields

`totalLength`
The length of the map entry, in bytes, from beginning of the record. This includes the fixed length, the length of the packed Pascal strings at the end of the entry, and the length of the user data that follow those strings.

`fixedLength`
The length of the fixed part of the map entry, in bytes, from beginning of the record.

`version`
The version number of the map entry. The only version currently defined is 0.

`fileType`
The four-character file type for the map entry.

`fileCreator`
The four-character creator code for the map entry.

`postCreator`
The creator code for the post-processing application for the map entry. Applications should consult this field only if the `ICmap_post_bit` is set in the `flags` field, but even if that bit is not set, applications can determine the post-processing application by looking at this field. If no post processing application has even been set, this field's value will be 0.

`flags`
: Flags associated with this map entry. For flag definitions, see "Map Constants" (page 2532), "Map Entry Flags" (page 2533), and "Map Entry Masks" (page 2534).

`extension`
: The filename extension for this map entry.

`creatorAppName`
: The name of the creator application identified by the `fileCreator` field. You can use this field to display the application name even if the application is not installed.

`postAppName`
: The name of the post-processing application identified by the `postCreator` field. You can use this field to display the post-processing application's name even if the that application is not installed. See the `post_creator` field description for details as to when this field is valid.

`MIMEType`
: The MIME type for the map entry, for example, "application/zip".

`entryName`
: The user-visible name for the map entry.

Discussion
The value of the Mappings preference is not an array, so you cannot index it directly. It is instead a packed array of `ICMapEntry` types, with each entry packed to remove the empty space at the end of the strings. Entries can start on an odd address and entries can contain user data . We strongly recommend that you access this data structure using the routines described in this section. These routines return an unpacked `ICMapEntry`, which is a lot easier to deal with.

Each `ICMapEntry` data type provides a filename extension, a MIME type, and a file type and creator. The database of `ICMapEntry` types is not normalised; that is, there can be multiple entries with the same creator, file type, MIME type, filename extension, and so on. In general, each application determines how these entries are used in a particular circumstance, although IC does define some policies by way of its high level mapping routines.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICServiceEntry

Specifies an Internet Config TCP service entry.

```
struct ICServiceEntry {
    Str255 name;
    short port;
    ICServiceEntryFlags flags;
};
typedef struct ICServiceEntry ICServiceEntry;
typedef ICServiceEntry * ICServiceEntryPtr;
typedef ICServiceEntryPtr * ICServiceEntryHandle;
```

Fields
`name`
: The name for the TCP service entry.

`port`
The port for the TCP service entry.

`flags`
For descriptions of the flags, see "Services Constants" (page 2535), "Services Bits" (page 2537), and "Services Masks" (page 2537).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

ICServices

Specifies the mapping between TCP service names and their ports.

```
struct ICServices {
    short count;
    ICServiceEntry services[1];
};
typedef struct ICServices ICServices;
typedef ICServices * ICServicesPtr;
typedef ICServicesPtr * ICServicesHandle;
```

Fields
`count`
`services`
An unbounded array of `ICServiceEntry` (page 2515)records.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`InternetConfig.h`

internetConfigurationComponent

```
typedef ComponentInstance internetConfigurationComponent;
```

Special Considerations

This data type is obsolete. Use the data type `ICInstance` (page 2513) instead.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`InternetConfig.h`

Constants

Apple Event Constants

```
enum {
    kInternetEventClass = 'GURL',
    kAEGetURL = 'GURL',
    kAEFetchURL = 'FURL',
    keyAEAttaching = 'Atch'
};
```

Attribute Constants

These are old names, mapped to newer names for backward compatibility, available only if you define `OLDROUTINENAMES`.

```
enum {
    ICattr_no_change = (unsigned long) (kICAttrNoChange),
    ICattr_locked_bit = kICAttrLockedBit,
    ICattr_locked_mask = kICAttrLockedMask,
    ICattr_volatile_bit = kICAttrVolatileBit,
    ICattr_volatile_mask = kICAttrVolatileMask,
    icNoUserInteraction_bit = kICNoUserInteractionBit,
    icNoUserInteraction_mask = kICNoUserInteractionMask,
    ICfiletype = kICFileType,
    ICcreator = kICCreator
};
```

Component Identifiers

Define the component type, subtype, and manufacturer of the Internet Config component.

```
enum {
    kICComponentType = 'PREF',
    kICComponentSubType = 'ICAp',
    kICComponentManufacturer = 'JPQE'
};
```

Component Identifiers (Deprecated)

```
enum {
    internetConfigurationComponentType = 'PREF',
    internetConfigurationComponentSubType = 'ICAp',
    internetConfigurationComponentInterfaceVersion0 = 0x00000000,
    internetConfigurationComponentInterfaceVersion1 = 0x00010000,
    internetConfigurationComponentInterfaceVersion2 = 0x00020000,
    internetConfigurationComponentInterfaceVersion3 = 0x00030000,
    internetConfigurationComponentInterfaceVersion =
internetConfigurationComponentInterfaceVersion3
};
```

Discussion
These constants are deprecated. Please use `Component Interface Version` (page 2518) and `Component Identifiers` (page 2517) instead.

Component Interface Version

Define the possible version numbers returned by the Internet Config component.

```
enum {
    kICComponentInterfaceVersion0 = 0x00000000,
    kICComponentInterfaceVersion1 = 0x00010000,
    kICComponentInterfaceVersion2 = 0x00020000,
    kICComponentInterfaceVersion3 = 0x00030000,
    kICComponentInterfaceVersion4 = 0x00040000,
    kICComponentInterfaceVersion = kICComponentInterfaceVersion4
};
```

Constants

`kICComponentInterfaceVersion0`
Internet Config versions greater than or equal to 1.0.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICComponentInterfaceVersion1`
Internet Config versions greater than or equal to 1.1.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICComponentInterfaceVersion2`
Internet Config versions greater than or equal to 1.2.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICComponentInterfaceVersion3`
Internet Config versions greater than or equal to 2.0.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICComponentInterfaceVersion4`
Internet Config versions greater than or equal to 2.5.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICComponentInterfaceVersion`
The current version number is 4.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Discussion
These constants define the possible version numbers returned by the Internet Config component in response to a `ICGetVersion` (page 2502) call. The constants define only the high word of the version number, which is the version of the programming interface. The low word of the version number is the implementation version, which changes with each bug fix release of Internet Config.

Component Selectors

Represent component selectors for each Internet Config function.

```
enum {
    kICCStart = 0,
    kICCStop = 1,
    kICCGetVersion = 50,
    kICCFindConfigFile = 2,
    kICCFindUserConfigFile = 14,
    kICCGeneralFindConfigFile = 30,
    kICCChooseConfig = 33,
    kICCChooseNewConfig = 34,
    kICCGetConfigName = 35,
    kICCGetConfigReference = 31,
    kICCSetConfigReference = 32,
    kICCSpecifyConfigFile = 3,
    kICCRefreshCaches = 47,
    kICCGetSeed = 4,
    kICCGetPerm = 13,
    kICCDefaultFileName = 11,
    kICCBegin = 5,
    kICCGetPref = 6,
    kICCSetPref = 7,
    kICCFindPrefHandle = 36,
    kICCGetPrefHandle = 26,
    kICCSetPrefHandle = 27,
    kICCCountPref = 8,
    kICCGetIndPref = 9,
    kICCDeletePref = 12,
    kICCEnd = 10,
    kICCGetDefaultPref = 49,
    kICCEditPreferences = 15,
    kICCLaunchURL = 17,
    kICCParseURL = 16,
    kICCCreateGURLEvent = 51,
    kICCSendGURLEvent = 52,
    kICCMapFilename = 24,
    kICCMapTypeCreator = 25,
    kICCMapEntriesFilename = 28,
    kICCMapEntriesTypeCreator = 29,
    kICCCountMapEntries = 18,
    kICCGetIndMapEntry = 19,
    kICCGetMapEntry = 20,
    kICCSetMapEntry = 21,
    kICCDeleteMapEntry = 22,
    kICCAddMapEntry = 23,
    kICCGetCurrentProfile = 37,
    kICCSetCurrentProfile = 38,
    kICCCountProfiles = 39,
    kICCGetIndProfile = 40,
    kICCGetProfileName = 41,
    kICCSetProfileName = 42,
    kICCAddProfile = 43,
    kICCDeleteProfile = 44,
    kICCRequiresInterruptSafe = 45,
    kICCGetMappingInterruptSafe = 46,
    kICCGetSeedInterruptSafe = 48,
    kICCFirstSelector = kICCStart,
    kICCLastSelector = 52
};
```

Discussion
These constants are component selectors for each Internet Config fuction. You can use these values as parameters to the Component Manager's `ComponentFunctionImplemented` function. These are also useful for authors of override components.

Component Selector Proc Information

```
enum {
    kICCStartProcInfo = 1008,
    kICCStopProcInfo = 240,
    kICCGetVersionProcInfo = 4080,
    kICCFindConfigFileProcInfo = 3824,
    kICCFindUserConfigFileProcInfo = 1008,
    kICCGeneralFindConfigFileProcInfo = 58864,
    kICCChooseConfigProcInfo = 240,
    kICCChooseNewConfigProcInfo = 240,
    kICCGetConfigNameProcInfo = 3568,
    kICCGetConfigReferenceProcInfo = 1008,
    kICCSetConfigReferenceProcInfo = 4080,
    kICCSpecifyConfigFileProcInfo = 1008,
    kICCRefreshCachesProcInfo = 240,
    kICCGetSeedProcInfo = 1008,
    kICCGetPermProcInfo = 1008,
    kICCDefaultFileNameProcInfo = 1008,
    kICCGetComponentInstanceProcInfo = 1008,
    kICCBeginProcInfo = 496,
    kICCGetPrefProcInfo = 65520,
    kICCSetPrefProcInfo = 65520,
    kICCFindPrefHandleProcInfo = 16368,
    kICCGetPrefHandleProcInfo = 16368,
    kICCSetPrefHandleProcInfo = 16368,
    kICCCountPrefProcInfo = 1008,
    kICCGetIndPrefProcInfo = 4080,
    kICCDeletePrefProcInfo = 1008,
    kICCEndProcInfo = 240,
    kICCGetDefaultPrefProcInfo = 4080,
    kICCEditPreferencesProcInfo = 1008,
    kICCLaunchURLProcInfo = 262128,
    kICCParseURLProcInfo = 1048560,
    kICCCreateGURLEventProcInfo = 16368,
    kICCSendGURLEventProcInfo = 1008,
    kICCMapFilenameProcInfo = 4080,
    kICCMapTypeCreatorProcInfo = 65520,
    kICCMapEntriesFilenameProcInfo = 16368,
    kICCMapEntriesTypeCreatorProcInfo = 262128,
    kICCCountMapEntriesProcInfo = 4080,
    kICCGetIndMapEntryProcInfo = 65520,
    kICCGetMapEntryProcInfo = 16368,
    kICCSetMapEntryProcInfo = 16368,
    kICCDeleteMapEntryProcInfo = 4080,
    kICCAddMapEntryProcInfo = 4080,
    kICCGetCurrentProfileProcInfo = 1008,
    kICCSetCurrentProfileProcInfo = 1008,
    kICCCountProfilesProcInfo = 1008,
    kICCGetIndProfileProcInfo = 4080,
    kICCGetProfileNameProcInfo = 4080,
    kICCSetProfileNameProcInfo = 4080,
    kICCAddProfileProcInfo = 4080,
    kICCDeleteProfileProcInfo = 1008,
    kICCRequiresInterruptSafeProcInfo = 240,
    kICCGetMappingInterruptSafeProcInfo = 4080,
    kICCGetSeedInterruptSafeProcInfo = 1008
};
```

Edit Preference Constants

Specify AppleEvent details for Internet Config preference requests.

```
enum {
    kICEditPreferenceEventClass = 'ICAp',
    kICEditPreferenceEvent = 'ICAp',
    keyICEditPreferenceDestination = 'dest'
};
```

Discussion
These constants specify details of the AppleEvent that the `ICEditPreferences` (page 2494) function sends to the Internet Config application to request it to open the edit window for a specific preference. Consult the Internet Config terminology resource (`'aete'`) for the meaning of these constants.

File Specification Header Size

```
enum {
    kICFileSpecHeaderSize = sizeof(ICFileSpec) - sizeof(AliasRecord)
};
```

File Specification Header Size (Deprecated)

```
enum {
    ICfile_spec_header_size = kICFileSpecHeaderSize
};
```

Discussion
This constant is available only if you define `OLDROUTINENAMES`.

File Type Constants

Define the file type and creator code for the Internet preferences file.

```
enum {
    kICFileType = 'ICAp',
    kICCreator = 'ICAp'
};
```

Constants

`kICFileType`
The Finder file type for the Internet preferences file.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICCreator`
The Finder creator code for the Internet preferences file.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Keys

```
#define kICReservedKey                  "\pkICReservedKey"
#define kICArchieAll                    "\pArchieAll"
#define kICArchiePreferred              "\pArchiePreferred"
#define kICCharacterSet                 "\pCharacterSet"
#define kICDocumentFont                 "\pDocumentFont"
#define kICDownloadFolder               "\pDownloadFolder"
#define kICEmail                        "\pEmail"
#define kICFTPHost                      "\pFTPHost"
#define kICFTPProxyAccount              "\pFTPProxyAccount"
#define kICFTPProxyHost                 "\pFTPProxyHost"
#define kICFTPProxyPassword             "\pFTPProxyPassword"
#define kICFTPProxyUser                 "\pFTPProxyUser"
#define kICFingerHost                   "\pFingerHost"
#define kICGopherHost                   "\pGopherHost"
#define kICGopherProxy                  "\pGopherProxy"
#define kICHTTPProxyHost                "\pHTTPProxyHost"
#define kICHelper                       "\pHelper•"
#define kICHelperDesc                   "\pHelperDesc•"
#define kICHelperList                   "\pHelperList•"
#define kICIRCHost                      "\pIRCHost"
#define kICInfoMacAll                   "\pInfoMacAll"
#define kICInfoMacPreferred             "\pInfoMacPreferred"
#define kICLDAPSearchbase               "\pLDAPSearchbase"
#define kICLDAPServer                   "\pLDAPServer"
#define kICListFont                     "\pListFont"
#define kICMacSearchHost                "\pMacSearchHost"
#define kICMailAccount                  "\pMailAccount"
#define kICMailHeaders                  "\pMailHeaders"
#define kICMailPassword                 "\pMailPassword"
#define kICMapping                      "\pMapping"
#define kICNNTPHost                     "\pNNTPHost"
#define kICNTPHost                      "\pNTPHost"
#define kICNewMailDialog                "\pNewMailDialog"
#define kICNewMailFlashIcon             "\pNewMailFlashIcon"
#define kICNewMailPlaySound             "\pNewMailPlaySound"
#define kICNewMailSoundName             "\pNewMailSoundName"
#define kICNewsAuthPassword             "\pNewsAuthPassword"
#define kICNewsAuthUsername             "\pNewsAuthUsername"
#define kICNewsHeaders                  "\pNewsHeaders"
#define kICNoProxyDomains               "\pNoProxyDomains"
#define kICOrganization                 "\pOrganization"
#define kICPhHost                       "\pPhHost"
#define kICPlan                         "\pPlan"
#define kICPrinterFont                  "\pPrinterFont"
#define kICQuotingString                "\pQuotingString"
#define kICRealName                     "\pRealName"
#define kICRTSPProxyHost                "\pRTSPProxyHost"
#define kICSMTPHost                     "\pSMTPHost"
#define kICScreenFont                   "\pScreenFont"
#define kICServices                     "\pServices"
#define kICSignature                    "\pSignature"
#define kICSnailMailAddress             "\pSnailMailAddress"
#define kICSocksHost                    "\pSocksHost"
#define kICTelnetHost                   "\pTelnetHost"
#define kICUMichAll                     "\pUMichAll"
#define kICUMichPreferred               "\pUMichPreferred"
```

```
#define kICUseFTPProxy                  "\pUseFTPProxy"
#define kICUseGopherProxy               "\pUseGopherProxy"
#define kICUseHTTPProxy                 "\pUseHTTPProxy"
#define kICUsePassiveFTP                "\pUsePassiveFTP"
#define kICUseRTSPProxy                 "\pUseRTSPProxy"
#define kICUseSocks                     "\pUseSocks"
#define kICWAISGateway                  "\pWAISGateway"
#define kICWWWHomePage                  "\pWWWHomePage"
#define kICWebBackgroundColour          "\pWebBackgroundColour"
#define kICWebReadColor                 "\p646F6777•WebReadColor"
#define kICWebSearchPagePrefs           "\pWebSearchPagePrefs"
#define kICWebTextColor                 "\pWebTextColor"
#define kICWebUnderlineLinks            "\p646F6777•WebUnderlineLinks"
#define kICWebUnreadColor               "\p646F6777•WebUnreadColor"
#define kICWhoisHost                    "\pWhoisHost"
```

Constants

`kICReservedKey`

Reserved for use by Internet Config.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICArchieAll`

Formatted string list of Archie servers.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICArchiePreferred`

Formatted string naming the preferred Archie server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICCharacterSet`

For conversion between kTextEncodingMacRoman (“Mac”) and kTextEncodingMacRomanLatin1 (“Net”). Deprecated. Use the Text Encoding Conversion manager and Unicode instead.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICDocumentFont`

Font to use for proportional text.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICDownloadFolder`

Location in the file system for newly downloaded files.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICEmail`

Return address for the user.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICFTPHost`
Default FTP server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICFTPProxyAccount`
Second-level FTP proxy authorization. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICFTPProxyHost`
Default FTP proxy server. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICFTPProxyPassword`
Password (scrambled) for FTP proxy user. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICFTPProxyUser`
First-level FTP proxy authorization. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICFingerHost`
Default finger-protocol server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICGopherHost`
Default gopher-protocol server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICGopherProxy`
Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICHTTPProxyHost`
Default HTTP proxy server. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICHelper`
Helpers for URL schemes.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICHelperDesc`
Descriptions for URL schemes.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICHelperList`
Common helpers for URL schemes.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICIRCHost`
Internet Relay Chat server.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICInfoMacAll`
Formatted list of Info-Mac servers.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICInfoMacPreferred`
Formatted string containing preferred Info-Mac server.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICLDAPSearchbase`
Default LDAP search base.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICLDAPServer`
Default LDAP server.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICListFont`
Font used for lists, such as news article lists.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMacSearchHost`
Host for MacSearch queries.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMailAccount`
Account from which to fetch email.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMailHeaders`
: Additional headers for email messages.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICMailPassword`
: Password (scrambled) for the default mail account.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICMapping`
: File-type mapping.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNNTPHost`
: Default NNTP server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNTPHost`
: Default NTP (Network Time Protocol) server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNewMailDialog`
: Boolean value indicating whether to display a new email dialog.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNewMailFlashIcon`
: Boolean value indicating whether to flash a new email icon.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNewMailPlaySound`
: Boolean value indicating whether to play a new email sound.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNewMailSoundName`
: The name of the preferred new email sound.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNewsAuthPassword`
: Password (scrambled) for authorized news account.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNewsAuthUsername`
User name for authorized news account.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNewsHeaders`
Additional headers for news messages.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICNoProxyDomains`
List of domains not to be proxied.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICOrganization`
String for the X-Organization MIME header.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICPhHost`
Default Ph protocol server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICPlan`
Default response for finger servers.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICPrinterFont`
Font used to print the monospaced text displayed on screen (see kICScreenFont).

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICQuotingString`
A short string used to indicate quoting in email and news replies.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICRealName`
Real name of user.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICRTSPProxyHost`
Default RTSP proxy server. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.1 and later.

Declared in `InternetConfig.h`.

`kICSMTPHost`
Default SMTP host.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICScreenFont`
Font for monospaced text.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICServices`
TCP and IP port-to-name mapping.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICSignature`
Block of text to append to outgoing news and email messages.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICSnailMailAddress`
Preferred postal mail address.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICSocksHost`
Default SOCKS host. (The host.domain format allows ":port" and " port")

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICTelnetHost`
Default Telnet host.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICUMichAll`
Formatted list of UMich servers.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICUMichPreferred`
Formatted string containing preferred UMich server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICUseFTPProxy`
Boolean value indicating whether to use an FTP proxy. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICUseGopherProxy`
: Boolean value indicating whether to use a Gopher proxy. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICUseHTTPProxy`
: Boolean value indicating whether to use an HTTP proxy. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICUsePassiveFTP`
: Boolean value indicating whether to use the PASV command for FTP transfers.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICUseRTSPProxy`
: Boolean value indicating whether to use an RTSP proxy. Read-only in Mac OS X; use System Configuration to set this value.

Available in Mac OS X v10.1 and later.

Declared in `InternetConfig.h`.

`kICUseSocks`
: Boolean value indicating whether to use SOCKS.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWAISGateway`
: Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWWWHomePage`
: User's default Web home page.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWebBackgroundColour`
: Background color for Web pages.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWebReadColor`
: Color for Web links that have been visited.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWebSearchPagePrefs`
: URL for the default search engine page.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWebTextColor`
Color for normal text on Web pages.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWebUnderlineLinks`
Boolean value indicating whether to underline Web links.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWebUnreadColor`
Color for Web links that have not been visited.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

`kICWhoisHost`
Default whois server.

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

Map Constants

```
enum {
    ICmap_binary_bit = kICMapBinaryBit,
    ICmap_binary_mask = kICMapBinaryMask,
    ICmap_resource_fork_bit = kICMapResourceForkBit,
    ICmap_resource_fork_mask = kICMapResourceForkMask,
    ICmap_data_fork_bit = kICMapDataForkBit,
    ICmap_data_fork_mask = kICMapDataForkMask,
    ICmap_post_bit = kICMapPostBit,
    ICmap_post_mask = kICMapPostMask,
    ICmap_not_incoming_bit = kICMapNotIncomingBit,
    ICmap_not_incoming_mask = kICMapNotIncomingMask,
    ICmap_not_outgoing_bit = kICMapNotOutgoingBit,
    ICmap_not_outgoing_mask = kICMapNotOutgoingMask,
    ICmap_fixed_length = kICMapFixedLength
};
```

Discussion
For descriptions of these constants, see "Map Entry Flags" (page 2533) and "Map Entry Masks" (page 2534). These constants are available only if you define `OLDROUTINENAMES`.

Map Entry Flags

```
typedef long ICMapEntryFlags;
enum {
    kICMapBinaryBit = 0,
    kICMapResourceForkBit = 1,
    kICMapDataForkBit = 2,
    kICMapPostBit = 3,
    kICMapNotIncomingBit = 4,
    kICMapNotOutgoingBit = 5
};
```

Constants

`kICMapBinaryBit`
When this bit is set, indicates the file should be transferred in binary as opposed to text mode.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapResourceForkBit`
When this bit is set, indicates the resource fork of the file is significant.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapDataForkBit`
When this bit is set, indicates the data fork of the file is significant.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapPostBit`
When this bit is set, indicates post process using post fields.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapNotIncomingBit`
When this bit is set, indicates to ignore this mapping for incoming files.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapNotOutgoingBit`
When this bit is set, indicates to ignore this mapping for outgoing files.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Map Entry Masks

```
enum {
    kICMapBinaryMask = 0x00000001,
    kICMapResourceForkMask = 0x00000002,
    kICMapDataForkMask = 0x00000004,
    kICMapPostMask = 0x00000008,
    kICMapNotIncomingMask = 0x00000010,
    kICMapNotOutgoingMask = 0x00000020
};
```

Constants

`kICMapBinaryMask`
Indicates the file should be transferred in binary as opposed to text mode.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapResourceForkMask`
Indicates the resource fork of the file is significant.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapDataForkMask`
Indicates the data fork of the file is significant.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapPostMask`
Indicates to post process using post fields.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapNotIncomingMask`
Indicates to ignore this mapping for incoming files.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICMapNotOutgoingMask`
Indicates to ignore this mapping for outgoing files.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Map Fixed Length Constants

```
typedef short     ICFixedLength;
enum {
    kICMapFixedLength = 22
};
```

Constants

`kICMapFixedLength`

Use in the `fixedLength` field of a structure `ICMapEntry` (page 2514).

Available in Mac OS X v10.0 and later.

Declared in `InternetConfig.h`.

Services Constants

```
enum {
    ICservices_tcp_bit = kICServicesTCPBit,
    ICservices_tcp_mask = kICServicesTCPMask,
    ICservices_udp_bit = kICServicesUDPBit,
    ICservices_udp_mask = kICServicesUDPMask
};
```

Special Considerations

These constants are available only if you define `OLDROUTINENAMES`.

Permissions

The ICPerm type is used to specify whether you wish to access the preferences as read-only or read/write.

```
typedef UInt8 ICPerm;
enum {
    icNoPerm = 0,
    icReadOnlyPerm = 1,
    icReadWritePerm = 2
};
```

Preference Attribute Bits and Masks

```
typedef UInt32 ICAttr;
enum {
    kICAttrLockedBit = 0,
    kICAttrVolatileBit = 1
};
```

```
enum {
    kICAttrNoChange = 0xFFFFFFFF,
    kICAttrLockedMask = 0x00000001,
    kICAttrVolatileMask = 0x00000002
};
```

Constants

`kICAttrLockedBit`
If the locked bit is set, any attempt to set the preference will result in an error.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICAttrVolatileBit`
If the volatile bit is set, you should not cache the value of this preference because it is subject to non-seed changing changes.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICAttrNoChange`
Used when you call ICSetPref and do not want to mess around with attributes. You can supply this value and IC will not change the attribute of the preference.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICAttrLockedMask`
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICAttrVolatileMask`
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Discussion
The ICAttr type is simply a longint containing flags that describe the attributes of a key and its data.

Preference Attribute Masks

```
enum {
    kICAttrNoChange = 0xFFFFFFFF,
    kICAttrLockedMask = 0x00000001,
    kICAttrVolatileMask = 0x00000002
};
```

Profile IDs

The `ICProfileID` type is an opaque reference to a particular profile.

```
typedef long ICProfileID;
typedef ICProfileID * ICProfileIDPtr;
enum {
    kICNilProfileID = 0
};
```

Constants

`kICNilProfileID`
Use to denote no profile.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Services Bits

```
typedef short ICServiceEntryFlags;
enum {
    kICServicesTCPBit = 0,
    kICServicesUDPBit = 1
};
```

Constants

`kICServicesTCPBit`
When this bit is set, indicates the service is TCP.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICServicesUDPBit`
When this bit is set, indicates the service is UDP.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Discussion
Both these bits may be set to indicate the service is both TCP and UDP.

Services Masks

```
enum {
    kICServicesTCPMask = 0x00000001,
    kICServicesUDPMask = 0x00000002
};
```

Constants

`kICServicesTCPMask`
Indicates the service is TCP.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

`kICServicesUDPMask`
Indicates the service is UDP.
Available in Mac OS X v10.0 and later.
Declared in `InternetConfig.h`.

Discussion
Both these bits may be set to indicate the service is both TCP and UDP.

User Interaction Constants

```
enum {
    kICNoUserInteractionBit = 0
};
```

User Interaction Masks

```
enum {
    kICNoUserInteractionMask = 0x00000001
};
```

Version Constants

```
enum {
    kICComponentVersion = 0,
    kICNumVersion = 1
};
```

Result Codes

The most common result codes returned by Internet Config are listed in the table below.

Result Code	Value	Description
`icPrefNotFoundErr`	-666	The preference was not found. Available in Mac OS X v10.0 and later.
`icPermErr`	-667	The preference cannot be set because of premissions restrictions. Available in Mac OS X v10.0 and later.
`icPrefDataErr`	-668	There was a problem with the preference data. Available in Mac OS X v10.0 and later.
`icInternalErr`	-669	An internal error occurred. Available in Mac OS X v10.0 and later.
`icTruncatedErr`	-670	More data was present than was returned. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
icNoMoreWritersErr	-671	You can't begin a write session because there is already one in progress. Available in Mac OS X v10.0 and later.
icNothingToOverrideErr	-672	There is no component for the override component to capture. Available in Mac OS X v10.0 and later.
icNoURLErr	-673	No URL was found. Available in Mac OS X v10.0 and later.
icConfigNotFoundErr	-674	No configuration was found. Available in Mac OS X v10.0 and later.
icConfigInappropriateErr	-675	The manufacturer code is incorrect. Available in Mac OS X v10.0 and later.
icProfileNotFoundErr	-676	The profile was not found. Available in Mac OS X v10.0 and later.
icTooManyProfilesErr	-677	There are too many profiles in the database. Available in Mac OS X v10.0 and later.

QuickDraw Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	QuickdrawAPI.h QuickdrawTypes.h QDOffscreen.h QDPictToCGContext.h
Companion guide	Quartz Programming Guide for QuickDraw Developers

Overview

QuickDraw is the legacy 2D drawing engine for Macintosh computers. QuickDraw provides routines for drawing, manipulating, and displaying graphic objects such as lines, arcs, rectangles, ovals, regions, and bitmap images. Carbon supports most of the classic QuickDraw programming interface.

Note: QuickDraw has been deprecated for deployment targets Mac OS X version 10.4 and later. The replacement API is Quartz 2D. Because of the fundamental differences in the imaging models and design goals between QuickDraw and Quartz, there is no direct correspondence between QuickDraw and Quartz concepts and interfaces. For certain purposes, some QuickDraw functions may still be needed during a transition period; nevertheless, most of them have been deprecated to express the overriding goal of eliminating the use of QuickDraw in the future.

Functions by Task

Drawing QuickDraw Pictures in a Quartz Context

`QDPictCreateWithProvider` (page 2766)
Creates a QDPict picture, using QuickDraw picture data supplied with a Quartz data provider.

`QDPictCreateWithURL` (page 2767)
Creates a QDPict picture, using QuickDraw picture data specified with a Core Foundation URL.

`QDPictDrawToCGContext` (page 2767)
Draws a QuickDraw picture in a Quartz context.

`QDPictGetBounds` (page 2768)
Returns the intended location and size of a QDPict picture.

`QDPictGetResolution` (page 2769)
Returns the horizontal and vertical resolution of a QDPict picture.

`QDPictRelease` (page 2769)
Releases a QDPict picture.

`QDPictRetain` (page 2770) Deprecated in Mac OS X v10.3
Retains a QDPict picture.

Using Quartz 2D to Draw in a Graphics Port

`QDBeginCGContext` (page 2756)
Returns a Quartz 2D drawing environment associated with a graphics port.

`QDEndCGContext` (page 2758)
Terminates a Quartz 2D drawing environment associated with a graphics port.

`ClipCGContextToRegion` (page 2579) Deprecated in Mac OS X v10.4
Sets the clipping path in a Quartz 2D graphics context, using a clipping region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`CreateCGContextForPort` (page 2591) Deprecated in Mac OS X v10.4
Creates a Quartz 2D drawing environment associated with a graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SyncCGContextOriginWithPort` (page 2826) Deprecated in Mac OS X v10.4
Synchronizes the origin in a Quartz context with the lower-left corner of the associated graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Other Quartz-Related Functions in QuickDraw

`QDGetCGDirectDisplayID` (page 2761)
Returns the Quartz display ID that corresponds to a QuickDraw graphics device.

`CreateNewPortForCGDisplayID` (page 2592) Deprecated in Mac OS X v10.4
Creates a graphics port associated with a display. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LockPortBits` (page 2705) Deprecated in Mac OS X v10.4
Acquires an exclusive lock on the back buffer for a Carbon window. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDFlushPortBuffer` (page 2760) Deprecated in Mac OS X v10.4
Calls the Quartz compositor to flush all new drawing in a Carbon window to the display. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`UnlockPortBits` (page 2829) Deprecated in Mac OS X v10.4
Releases a previously acquired lock on the back buffer for a Carbon window. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Calculating Black-and-White Fills

`CalcMask` (page 2578) Deprecated in Mac OS X v10.4
Determines where filling will not occur when filling from the outside of a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SeedFill` (page 2788) Deprecated in Mac OS X v10.4
Determines how far filling will extend from a seeding point. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Calculating Color Fills

`CalcCMask` (page 2577) Deprecated in Mac OS X v10.4
Determines where filling will not occur when filling from the outside of a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SeedCFill` (page 2787) Deprecated in Mac OS X v10.4
Determines how far filling will extend to pixels matching the color of a particular pixel. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing Black-and-White Cursors

`GetCursor` (page 2638) Deprecated in Mac OS X v10.4
Loads a cursor resource into memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetCursor` (page 2792) Deprecated in Mac OS X v10.4
Sets the current cursor. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing Color Cursors

`AllocCursor` (page 2572) Deprecated in Mac OS X v10.4
Reallocates cursor memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeCCursor` (page 2598) Deprecated in Mac OS X v10.4
Disposes of all structures allocated by the `GetCCursor` function. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetCCursor` (page 2635) Deprecated in Mac OS X v10.4
Loads a color cursor resource into memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetCCursor` (page 2790) Deprecated in Mac OS X v10.4
Specifies a color cursor for display on the screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing the Background Bit Pattern

`BackPat` (page 2575) Deprecated in Mac OS X v10.4
Changes the bit pattern used as the background pattern by the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Changing the Background Pixel Pattern

`BackPixPat` (page 2576) Deprecated in Mac OS X v10.4
Assigns a pixel pattern as the background pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Compressing and Decompressing Data

`PackBits` (page 2739) Deprecated in Mac OS X v10.4
Compresses a data buffer stored in RAM. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`UnpackBits` (page 2830) Deprecated in Mac OS X v10.4
Decompresses a data buffer containing data compressed by `PackBits`. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Converting Between Angle and Slope Values

`AngleFromSlope` (page 2574)
Converts a slope value to an angle value.

`SlopeFromAngle` (page 2814)
Converts an angle value to a slope value.

Copying Images

`CopyBits` (page 2584) Deprecated in Mac OS X v10.4
Copies a portion of a bitmap or a pixel map from one graphics port or offscreen graphics world into another graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`CopyDeepMask` (page 2586) Deprecated in Mac OS X v10.4
Uses a mask when copying bitmaps or pixel maps between graphics ports (or from an offscreen graphics world into a graphics port). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`CopyMask` (page 2588) Deprecated in Mac OS X v10.4
Copies a bit or pixel image from one graphics port or offscreen graphics world into another graphics port only where the bits in a mask are set to 1. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating, Altering, and Disposing of Offscreen Graphics Worlds

`NewGWorld` (page 2715)
Creates an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeGWorld` (page 2601) Deprecated in Mac OS X v10.4
Disposes of all the memory allocated for an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposeScreenBuffer` (page 2609) Deprecated in Mac OS X v10.4
Disposes an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewScreenBuffer` (page 2727) Deprecated in Mac OS X v10.4
Creates an offscreen `PixMap` structure and allocates memory for the base address of its pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewTempScreenBuffer` (page 2728) Deprecated in Mac OS X v10.4
Creates an offscreen `PixMap` structure and allocate temporary memory for the base address of its pixel image applications generally don't need to use `NewTempScreenBuffer`. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`UpdateGWorld` (page 2831) Deprecated in Mac OS X v10.4
Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Disposing of Color Tables

`DisposeCTable` (page 2599) Deprecated in Mac OS X v10.4
Disposes a `ColorTable` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetCTable` (page 2636) Deprecated in Mac OS X v10.4
Obtains a color table stored in a `'clut'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Disposing of Pictures

`ClosePicture` (page 2581) Deprecated in Mac OS X v10.4
Completes the collection of drawing commands and picture comments that define your picture. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`KillPicture` (page 2691) Deprecated in Mac OS X v10.4
Releases the memory occupied by a picture not stored in a `'PICT'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpenCPicture` (page 2734) Deprecated in Mac OS X v10.4
Begins defining a picture in extended version 2 format. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpenPicture` (page 2736) Deprecated in Mac OS X v10.4
Creates a picture which allows you to specify resolutions for your pictures. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PicComment` (page 2748) Deprecated in Mac OS X v10.4
Inserts a picture comment into a picture that you are defining or into your printing code. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Disposing of Pixel Patterns

`CopyPixPat` (page 2590) Deprecated in Mac OS X v10.4
Copies the contents of one pixel pattern to another. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposePixPat` (page 2602) Deprecated in Mac OS X v10.4
Releases the storage allocated to a pixel pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPixPat` (page 2651) Deprecated in Mac OS X v10.4
Obtains a pixel pattern (`'ppat'`) resource stored in a resource file. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MakeRGBPat` (page 2706) Deprecated in Mac OS X v10.4
Creates the appearance of otherwise unavailable colors on indexed devices. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewPixPat` (page 2720) Deprecated in Mac OS X v10.4
Creates a new pixel pattern. Generally, however, your application should create a pixel pattern in a `'ppat'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Managing Polygons

`ClosePoly` (page 2581) Deprecated in Mac OS X v10.4
Completes the collection of lines that defines a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`KillPoly` (page 2691) Deprecated in Mac OS X v10.4
Releases the memory occupied by a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OffsetPoly` (page 2731) Deprecated in Mac OS X v10.4
Moves a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpenPoly` (page 2737) Deprecated in Mac OS X v10.4
Begins defining a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating and Managing Rectangles

`EmptyRect` (page 2611)
Determines whether a rectangle is an empty rectangle.

`EqualRect` (page 2613)
Determines whether two rectangles are equal.

`InsetRect` (page 2672)
Shrinks or expands a rectangle.

`OffsetRect` (page 2732)
Moves a rectangle.

`Pt2Rect` (page 2753)
Determines the smallest rectangle that encloses two given points.

`PtInRect` (page 2753)
Determines whether a pixel below is enclosed in a rectangle.

`PtToAngle` (page 2754)
Calculates an angle between a vertical line pointing straight up from the center of a rectangle and a line from the center to a given point.

`SectRect` (page 2785)
Determines whether two rectangles intersect.

`SetRect` (page 2809)
Assigns coordinates to a rectangle.

`UnionRect` (page 2827)
Calculates the smallest rectangle that encloses two rectangles.

Creating and Managing Regions

`CopyRgn` (page 2590)
Makes a copy of a region.

`DiffRgn` (page 2598)
Subtracts one region from another.

`DisposeRgn` (page 2609)
Releases the memory occupied by a region.

`EmptyRgn` (page 2612)
Determines whether a region is empty.

`EqualRgn` (page 2614)
Determines whether two regions have identical sizes, shapes, and locations.

`InsetRgn` (page 2673)
Shrinks or expands a region.

`NewRgn` (page 2726)
Begins creating a new region.

`OffsetRgn` (page 2732)
Moves a region.

`PtInRgn` (page 2754)
Determines whether a pixel is within a region.

`RectInRgn` (page 2776)
Determines whether a rectangle intersects a region.

`RectRgn` (page 2776)
Changes the structure of an existing region to that of a rectangle.

`SectRgn` (page 2786)
Calculates the intersection of two regions.

`SetEmptyRgn` (page 2794)
Sets an existing region to be empty.

`SetRectRgn` (page 2810)
Changes the structure of an existing region to that of a rectangle.

`UnionRgn` (page 2828)
Calculates the union of two regions.

`XorRgn` (page 2833)
Calculates the difference between the union and the intersection of two regions.

`CloseRgn` (page 2582) Deprecated in Mac OS X v10.4
Organizes a collection of lines and shapes into a region definition. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpenRgn` (page 2737) Deprecated in Mac OS X v10.4
Begins defining a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating, Setting, and Disposing of GDevice Records

`DisposeGDevice` (page 2601) Deprecated in Mac OS X v10.4
Disposes of a `GDevice` structure, releases the space allocated for it, and disposes of all the data structures allocated for it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InitGDevice` (page 2671) Deprecated in Mac OS X v10.4
Initializes a `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewGDevice` (page 2714) Deprecated in Mac OS X v10.4
Creates a new `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetDeviceAttribute` (page 2793) Deprecated in Mac OS X v10.4
Sets the attribute bits of a `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetGDevice` (page 2796) Deprecated in Mac OS X v10.4
Sets a `GDevice` structure as the current device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Creating, Setting, and Disposing of Pixel Maps

`CopyPixMap` (page 2589) Deprecated in Mac OS X v10.4
Duplicates a `PixMap` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DisposePixMap` (page 2602) Deprecated in Mac OS X v10.4
Disposes a `PixMap` structure and its color table. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NewPixMap` (page 2719) Deprecated in Mac OS X v10.4
Creates a new, initialized `PixMap` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPortPix` (page 2805) Deprecated in Mac OS X v10.4
Sets the pixel map for the current color graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Customizing Color QuickDraw Operations

`SetStdCProcs` (page 2811) Deprecated in Mac OS X v10.4
Obtains a `CQDProcs` structure with fields that point to QuickDraw's standard low-level functions, which you can modify to change QuickDraw's standard low-level behavior. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Customizing QuickDraw Operations

`SetStdProcs` (page 2812) Deprecated in Mac OS X v10.4
Obtains a `QDProcs` structure with fields that point to basic QuickDraw's standard low-level functions, which you can modify to point to your own functions. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdArc` (page 2815) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for drawing an arc or a wedge. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdBits` (page 2816) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for transferring bits and pixels. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdComment` (page 2817) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for processing a picture comment. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdGetPic` (page 2817) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for retrieving information from the definition of a picture. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdLine` (page 2818) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for drawing a line. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdOval` (page 2819) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for drawing an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdPoly` (page 2820) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for drawing a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdPutPic` (page 2820) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for saving information as the definition of a picture. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdRect` (page 2821) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for drawing a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdRgn` (page 2822) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for drawing a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StdRRect` (page 2822) Deprecated in Mac OS X v10.4
QuickDraw's standard low-level function for drawing a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Determining Current Colors and Best Intermediate Colors

`GetBackColor` (page 2634) Deprecated in Mac OS X v10.4
Obtains the background color of the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetCPixel` (page 2636) Deprecated in Mac OS X v10.4
Determines the color of an individual pixel specified in the `h` and `v` parameters. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetForeColor` (page 2639) Deprecated in Mac OS X v10.4
Obtains the color of the foreground color for the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Determining the Characteristics of a Video Device

`DeviceLoop` (page 2597) Deprecated in Mac OS X v10.4
Draws images that are optimized for every screen they cross. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ScreenRes` (page 2783) Deprecated in Mac OS X v10.4
Determines the resolution of the main device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`TestDeviceAttribute` (page 2826) Deprecated in Mac OS X v10.4
Determines whether the flag bit for an attribute has been set in the `gdFlags` field of a `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Determining Whether QuickDraw Has Finished Drawing

`QDDone` (page 2758) Deprecated in Mac OS X v10.4
Determines whether QuickDraw has completed drawing in a given graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Arcs and Wedges

`EraseArc` (page 2615) Deprecated in Mac OS X v10.4
Erases a wedge. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillArc` (page 2619) Deprecated in Mac OS X v10.4
Fills a wedge with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameArc` (page 2629) Deprecated in Mac OS X v10.4
Draws an arc of the oval that fits inside a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertArc` (page 2674) Deprecated in Mac OS X v10.4
Inverts the pixels of a wedge. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintArc` (page 2740) Deprecated in Mac OS X v10.4
Paints a wedge of the oval that fits inside a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Lines

`Line` (page 2692) Deprecated in Mac OS X v10.4
Draws a line a specified distance from the graphics pen's current location in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LineTo` (page 2693) Deprecated in Mac OS X v10.4
Draws a line from the graphics pen's current location to a new location. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`Move` (page 2711) Deprecated in Mac OS X v10.4
Moves the graphics pen a particular distance. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MoveTo` (page 2712) Deprecated in Mac OS X v10.4
Moves the graphics pen to a particular location in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Ovals

`EraseOval` (page 2615) Deprecated in Mac OS X v10.4
Erases an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillOval` (page 2624) Deprecated in Mac OS X v10.4
Fills an oval with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameOval` (page 2630) Deprecated in Mac OS X v10.4
Draws an outline inside an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertOval` (page 2675) Deprecated in Mac OS X v10.4
Inverts the pixels enclosed by an oval. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintOval` (page 2741) Deprecated in Mac OS X v10.4
Paints an oval with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Pictures

`DrawPicture` (page 2610) Deprecated in Mac OS X v10.4
Draws a picture on any type of output device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPicture` (page 2648) Deprecated in Mac OS X v10.4
Obtains a handle to a picture stored in a `'PICT'` resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Polygons

`ErasePoly` (page 2616) Deprecated in Mac OS X v10.4
Erases a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillPoly` (page 2625) Deprecated in Mac OS X v10.4
Fills a polygon with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FramePoly` (page 2630) Deprecated in Mac OS X v10.4
Draws the outline of a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertPoly` (page 2676) Deprecated in Mac OS X v10.4
Inverts the pixels enclosed by a polygon. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintPoly` (page 2741) Deprecated in Mac OS X v10.4
Paints a polygon with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Rectangles

`EraseRect` (page 2617) Deprecated in Mac OS X v10.4
Erases a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillRect` (page 2626) Deprecated in Mac OS X v10.4
Fills a rectangle with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameRect` (page 2631) Deprecated in Mac OS X v10.4
Draws an outline inside a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertRect` (page 2677) Deprecated in Mac OS X v10.4
Inverts the pixels enclosed by a rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintRect` (page 2742) Deprecated in Mac OS X v10.4
Paints a rectangle with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Regions

`EraseRgn` (page 2618) Deprecated in Mac OS X v10.4
Erases a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillRgn` (page 2626) Deprecated in Mac OS X v10.4
Fills a region with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameRgn` (page 2632) Deprecated in Mac OS X v10.4
Draws an outline inside a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertRgn` (page 2678) Deprecated in Mac OS X v10.4
Inverts the pixels enclosed by a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintRgn` (page 2743) Deprecated in Mac OS X v10.4
Paints a region with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing Rounded Rectangles

`EraseRoundRect` (page 2618) Deprecated in Mac OS X v10.4
Erases a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillRoundRect` (page 2627) Deprecated in Mac OS X v10.4
Fills a rounded rectangle with any available bit pattern. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FrameRoundRect` (page 2633) Deprecated in Mac OS X v10.4
Draws an outline inside a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertRoundRect` (page 2679) Deprecated in Mac OS X v10.4
Inverts the pixels enclosed by a rounded rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PaintRoundRect` (page 2743) Deprecated in Mac OS X v10.4
Paints a rounded rectangle with the graphics pen's pattern and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing With Color QuickDraw Colors

`FillCArc` (page 2620) Deprecated in Mac OS X v10.4
Fills a wedge with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCOval` (page 2621) Deprecated in Mac OS X v10.4
Fills an oval with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCPoly` (page 2621) Deprecated in Mac OS X v10.4
Fills a polygon with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCRect` (page 2622) Deprecated in Mac OS X v10.4
Fills a rectangle with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCRgn` (page 2622) Deprecated in Mac OS X v10.4
Fills a region with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`FillCRoundRect` (page 2623) Deprecated in Mac OS X v10.4
Fills a rounded rectangle with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`HiliteColor` (page 2670) Deprecated in Mac OS X v10.4
Changes the highlight color for the current color graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`OpColor` (page 2733) Deprecated in Mac OS X v10.4
Sets the maximum color values for the `addPin` and `subPin` arithmetic transfer modes and the weight color for the `blend` arithmetic transfer mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RGBBackColor` (page 2779) Deprecated in Mac OS X v10.4
Changes the background color. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RGBForeColor` (page 2780) Deprecated in Mac OS X v10.4
Changes the color of the "ink" used for framing and painting. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetCPixel` (page 2791) Deprecated in Mac OS X v10.4
Sets the color of an individual pixel to the color that most closely matches the RGB color that you specify in the `cPix` parameter. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Drawing With the Eight-Color System

`BackColor` (page 2574) Deprecated in Mac OS X v10.4
Changes a basic graphics port's background color. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ColorBit` (page 2584) Deprecated in Mac OS X v10.4
Sets the foreground color for all printing in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ForeColor` (page 2628) Deprecated in Mac OS X v10.4
Changes the color of the "ink" used for framing, painting, and filling on computers that support only basic QuickDraw. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Getting Pattern Resources

`GetIndPattern` (page 2643) Deprecated in Mac OS X v10.4
Obtains a pattern stored in a pattern list (`'PAT#'`) resource. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPattern` (page 2646) Deprecated in Mac OS X v10.4
Obtains a pattern (`'PAT'`) resource stored in a resource file. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Getting the Available Graphics Devices

`GetDeviceList` (page 2639) Deprecated in Mac OS X v10.4
: Obtains a handle to the first `GDevice` structure in the device list. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetGDevice` (page 2640) Deprecated in Mac OS X v10.4
: Obtains a handle to the `GDevice` structure for the current device. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetMainDevice` (page 2644) Deprecated in Mac OS X v10.4
: Obtains a handle to the `GDevice` structure for the main screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetMaxDevice` (page 2644) Deprecated in Mac OS X v10.4
: Obtains a handle to the `GDevice` structure for the video device with the greatest pixel depth. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetNextDevice` (page 2645) Deprecated in Mac OS X v10.4
: Returns a handle to the next `GDevice` structure in the device list. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Hiding and Showing Cursors

`InitCursor` (page 2671)
: Sets the cursor to the standard arrow and makes the cursor visible.

`HideCursor` (page 2668)
: Hides the cursor if it is visible on the screen.

`ObscureCursor` (page 2730)
: Hides the cursor until the next time the user moves the mouse.

`ShieldCursor` (page 2813)
: Hides the cursor in a rectangle.

`ShowCursor` (page 2813)
: Displays a cursor hidden by the `HideCursor` or `ShieldCursor` functions.

Managing a Color Graphics Pen

`PenPixPat` (page 2747) Deprecated in Mac OS X v10.4
: Sets the pixel pattern used by the graphics pen in the current color graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing an Offscreen Graphics World's Pixel Image

`GetPixBaseAddr` (page 2648)
: Obtains a pointer to an offscreen pixel map.

`AllowPurgePixels` (page 2573) Deprecated in Mac OS X v10.4
: Makes the base address for an offscreen pixel image purgeable. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetGWorldPixMap` (page 2642) Deprecated in Mac OS X v10.4
Obtains the pixel map created for an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPixelsState` (page 2651) Deprecated in Mac OS X v10.4
Saves the current information about the memory allocated for an offscreen pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LockPixels` (page 2704) Deprecated in Mac OS X v10.4
Prevents the base address for an offscreen pixel image from being moved while you draw into or copy from its pixel map. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`NoPurgePixels` (page 2729) Deprecated in Mac OS X v10.4
Prevents the Memory Manager from purging the base address for an offscreen pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PixMap32Bit` (page 2749) Deprecated in Mac OS X v10.4
Determines whether a pixel map requires 32-bit addressing mode for access to its pixel image. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPixelsState` (page 2799) Deprecated in Mac OS X v10.4
Restores an offscreen pixel image to the state that you saved with the `GetPixelsState` function. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`UnlockPixels` (page 2829) Deprecated in Mac OS X v10.4
Allows the Memory Manager to move the base address for the offscreen pixel map that you specify in the `pm` parameter. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing Bitmaps, Port Rectangles, and Clipping Regions

`GetClip` (page 2635)
Saves the clipping region of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetClip` (page 2791)
Changes the clipping region of the current graphics port (basic or color) to a region you specify. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`BitMapToRegion` (page 2576) Deprecated in Mac OS X v10.4
Converts a bitmap or pixel map to a region. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ClipRect` (page 2580) Deprecated in Mac OS X v10.4
Changes the clipping region of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MovePortTo` (page 2711) Deprecated in Mac OS X v10.4
Changes the position of the port rectangle of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PortSize` (page 2751) Deprecated in Mac OS X v10.4
Changes the size of the port rectangle of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ScrollRect` (page 2784) Deprecated in Mac OS X v10.4
Scroll the pixels of a specified portion of a basic graphics port's bitmap (or a color graphics port's pixel map). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetOrigin` (page 2797) Deprecated in Mac OS X v10.4
Changes the coordinates of the window origin of the port rectangle of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPortBits` (page 2801) Deprecated in Mac OS X v10.4
Sets the bitmap for the current basic graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing Color Tables

`GetCTSeed` (page 2638) Deprecated in Mac OS X v10.4
Obtains a unique seed value for a color table created by your application. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ProtectEntry` (page 2752) Deprecated in Mac OS X v10.4
Adds protection to or removes protection from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ReserveEntry` (page 2777) Deprecated in Mac OS X v10.4
Reserves or removes reservation from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RestoreEntries` (page 2778) Deprecated in Mac OS X v10.4
Restores a selection of color table entries. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SaveEntries` (page 2781) Deprecated in Mac OS X v10.4
Saves a selection of color table entries. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetEntries` (page 2794) Deprecated in Mac OS X v10.4
Sets a group of color table entries for the current `GDevice` data structure. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing Colors

`Color2Index` (page 2583) Deprecated in Mac OS X v10.4
Obtains the index of the best available approximation for a given color in the color table of the current `GDevice` data structure. This function is used only by system software. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetSubTable` (page 2667) Deprecated in Mac OS X v10.4

Searches one color table for the best matches to colors in another color table. Your application should not need to call this function; it is used by system software only. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`Index2Color` (page 2670) Deprecated in Mac OS X v10.4

Obtains the `RGBColor` data structure corresponding to an index value in the color table of the current `GDevice` data structure. Your application should not need to call this function; it is used by system software only. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`InvertColor` (page 2675) Deprecated in Mac OS X v10.4

Finds the complement of an `RGBColor` data structure. This function is used only by system software. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`MakeITable` (page 2706) Deprecated in Mac OS X v10.4

Generates an inverse table for a color table. Your application should not need to call this function; it is used by system software only. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`RealColor` (page 2775) Deprecated in Mac OS X v10.4

Determines whether a given `RGBColor` data structure exists in the current device's color table. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Managing the Graphics Pen

`GetPen` (page 2646) Deprecated in Mac OS X v10.4

Determines the location of the graphics pen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPenState` (page 2647) Deprecated in Mac OS X v10.4

Determines the graphics pen's location, size, pattern, and pattern mode. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`HidePen` (page 2669) Deprecated in Mac OS X v10.4

Makes the graphics pen invisible, so that pen drawing doesn't show on the screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenMode` (page 2744) Deprecated in Mac OS X v10.4

Sets the pattern mode of the graphics pen in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenNormal` (page 2745) Deprecated in Mac OS X v10.4

Sets the size, pattern, and pattern mode of the graphics pen in the current graphics port to their initial values. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenPat` (page 2746) Deprecated in Mac OS X v10.4

Sets the bit pattern to be used by the graphics pen in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PenSize` (page 2747) Deprecated in Mac OS X v10.4

Sets the dimensions of the graphics pen in the current graphics port. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPenState` (page 2798) Deprecated in Mac OS X v10.4

Restores the state of the graphics pen that was saved with the `GetPenState` function. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`ShowPen` (page 2814) Deprecated in Mac OS X v10.4
Changes the ink of a graphics pen from invisible to visible, making pen drawing appear on the screen. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Manipulating Points in Graphics Ports

`AddPt` (page 2571)
Adds the coordinates of two points.

`EqualPt` (page 2613)
Determines whether the coordinates of two given points are equal.

`SetPt` (page 2807)
Assigns two coordinates to a point.

`SubPt` (page 2824)
Subtracts the coordinates of one point from another.

`DeltaPoint` (page 2596) Deprecated in Mac OS X v10.4
Subtracts the coordinates of one point from another. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetPixel` (page 2650) Deprecated in Mac OS X v10.4
Determines whether the pixel associated with a point is black or white. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GlobalToLocal` (page 2667) Deprecated in Mac OS X v10.4
Converts the coordinates of a point from global coordinates to the local coordinates of the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`LocalToGlobal` (page 2703) Deprecated in Mac OS X v10.4
Converts a point's coordinates from the local coordinates of the current graphics port (basic or color) to global coordinates. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Obtaining a Pseudorandom Number

`Random` (page 2775) Deprecated in Mac OS X v10.4
Obtains a pseudorandom integer. (Deprecated. Use the Standard C Library `random(3)` function instead.)

Operations on Search and Complement Functions

`AddComp` (page 2571) Deprecated in Mac OS X v10.4
Adds a function to the head of the current device data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`AddSearch` (page 2572) Deprecated in Mac OS X v10.4
Adds a function to the head of the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DelComp` (page 2595) Deprecated in Mac OS X v10.4
Removes a custom complement function from the current `GDevice` data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`DelSearch` (page 2595) Deprecated in Mac OS X v10.4
Removes a custom search function from the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetClientID` (page 2790) Deprecated in Mac OS X v10.4
Sets the `gdID` field in the current `GDevice` data structure to identify this client program to its search and complement functions. This function is used by system software and your application should not need to call it. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Reporting Data Structure Changes to QuickDraw

`CTabChanged` (page 2593) Deprecated in Mac OS X v10.4
Signals QuickDraw that the content of a `ColorTable` structure has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GDeviceChanged` (page 2633) Deprecated in Mac OS X v10.4
Notifies QuickDraw that the content of a `GDevice` structure has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PixPatChanged` (page 2750) Deprecated in Mac OS X v10.4
Notifies QuickDraw that the content of a `PixPat` structure, including its `PixMap` structure or the image in its `patData` field, has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`PortChanged` (page 2751) Deprecated in Mac OS X v10.4
Notifies QuickDraw that the content of a `GrafPort` structure or `CGrafPort` structure, including any of the data structures specified by handles within the structure, has been modified. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Retrieving Color QuickDraw Result Codes

`QDError` (page 2759) Deprecated in Mac OS X v10.4
Obtains a result code from the last applicable QuickDraw function that you called. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Saving and Restoring Graphics Ports

`GetPort` (page 2652) Deprecated in Mac OS X v10.4
Saves the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetPort` (page 2799) Deprecated in Mac OS X v10.4
Changes the current graphics port (basic or color). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

`GetGWorld` (page 2640)
Saves the current graphics port (basic, color, or offscreen) and the current `GDevice` structure. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SetGWorld` (page 2796)
Changes the current graphics port (basic, color, or offscreen). (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`GetGWorldDevice` (page 2641) Deprecated in Mac OS X v10.4
Obtains a handle to the `GDevice` structure associated with an offscreen graphics world. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Scaling and Mapping Points, Rectangles, Polygons, and Regions

`MapPt` (page 2708)
Maps a point in one rectangle to an equivalent position in another rectangle.

`MapRect` (page 2709)
Maps and scales a rectangle within one rectangle to another rectangle.

`MapRgn` (page 2710)
Maps and scales a region within one rectangle to another rectangle.

`ScalePt` (page 2782)
Scales a height and width according to the proportions of two rectangles.

`MapPoly` (page 2707) Deprecated in Mac OS X v10.4
Maps and scales a polygon within one rectangle to another rectangle. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Miscellaneous

`GetRegionBounds` (page 2666)

`HandleToRgn` (page 2668)

`IsRegionRectangular` (page 2690)

`IsValidRgnHandle` (page 2690)

`QDGetPatternOrigin` (page 2762)
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`QDRegionToRects` (page 2770)

`RgnToHandle` (page 2781)

`CloseCursorComponent` (page 2581) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

CreateNewPort (page 2592) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

CursorComponentChanged (page 2594) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

CursorComponentSetData (page 2594) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

deltapoint (page 2596) Deprecated in Mac OS X v10.4
(Deprecated. Use DeltaPoint (page 2596) instead.)

DisposeColorComplementUPP (page 2599) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeColorSearchUPP (page 2599) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeDeviceLoopDrawingUPP (page 2600) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeDragGrayRgnUPP (page 2600) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposePort (page 2603) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDArcUPP (page 2603) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDBitsUPP (page 2603) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDCommentUPP (page 2604) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDGetPicUPP (page 2604) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDJShieldCursorUPP (page 2604) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDLineUPP (page 2605) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDOpcodeUPP (page 2605) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDOvalUPP (page 2605) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDPolyUPP (page 2606) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDPutPicUPP (page 2606) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDRectUPP (page 2606) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDRgnUPP (page 2607) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDRRectUPP (page 2607) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDStdGlyphsUPP (page 2607) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDTextUPP (page 2608) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeQDTxMeasUPP (page 2608) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

DisposeRegionToRectsUPP (page 2608) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetMaskTable (page 2644) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPixBounds (page 2649) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPixDepth (page 2650) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPixRowBytes (page 2652) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBackColor (page 2653) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBackPixPat (page 2653) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBitMapForCopyBits (page 2654) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortBounds (page 2654) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortChExtra (page 2655) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortClipRegion (page 2655) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortCustomXFerProc (page 2655) Deprecated in Mac OS X v10.4

GetPortFillPixPat (page 2656) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortForeColor (page 2656) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortFracHPenLocation (page 2657) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortGrafProcs (page 2657) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortHiliteColor (page 2658) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortOpColor (page 2658) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenLocation (page 2658) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenMode (page 2659) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenPixPat (page 2659) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenSize (page 2660) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPenVisibility (page 2660) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortPixMap (page 2660) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortSpExtra (page 2661) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextFace (page 2661) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextFont (page 2662) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextMode (page 2662) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortTextSize (page 2662) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetPortVisibleRegion (page 2663) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsArrow (page 2663) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsBlack (page 2663) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsDarkGray (page 2664) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsGray (page 2664) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsLightGray (page 2665) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsRandomSeed (page 2665) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsScreenBits (page 2665) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsThePort (page 2666) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GetQDGlobalsWhite (page 2666) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

GrafDevice (page 2668) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeColorComplementUPP (page 2680) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeColorSearchUPP (page 2680) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeDeviceLoopDrawingUPP (page 2680) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeDragGrayRgnUPP (page 2681) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDArcUPP (page 2681) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDBitsUPP (page 2682) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDCommentUPP (page 2682) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDGetPicUPP (page 2682) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDJShieldCursorUPP (page 2683) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDLineUPP (page 2683) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDOpcodeUPP (page 2683) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDOvalUPP (page 2684) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDPolyUPP (page 2684) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDPutPicUPP (page 2685) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDRectUPP (page 2685) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDRgnUPP (page 2685) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDRRectUPP (page 2686) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDStdGlyphsUPP (page 2686) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDTextUPP (page 2686) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeQDTxMeasUPP (page 2687) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

InvokeRegionToRectsUPP (page 2687) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortClipRegionEmpty (page 2687) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortColor (page 2688) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortOffscreen (page 2688) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortPictureBeingDefined (page 2688) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortPolyBeingDefined (page 2689) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortRegionBeingDefined (page 2689) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsPortVisibleRegionEmpty (page 2689) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

IsValidPort (page 2690) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.5
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetCursorNew (page 2693) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetDeviceList (page 2694) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetFractEnable (page 2694) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetHiliteMode (page 2694) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetHiliteRGB (page 2695) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetLastFOND (page 2695) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetLastSPExtra (page 2695) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetMainDevice (page 2696) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetQDColors (page 2696) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetScrHRes (page 2696) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetScrVRes (page 2697) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetTheGDevice (page 2697) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetWidthListHand (page 2697) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetWidthPtr (page 2698) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMGetWidthTabHandle (page 2698) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetCursorNew (page 2698) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetDeviceList (page 2699) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetFractEnable (page 2699) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetHiliteMode (page 2699) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetHiliteRGB (page 2700) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetLastFOND (page 2700) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetLastSPExtra (page 2700) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetMainDevice (page 2701) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetQDColors (page 2701) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetScrHRes (page 2701) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetScrVRes (page 2702) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetTheGDevice (page 2702) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetWidthListHand (page 2702) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetWidthPtr (page 2703) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

LMSetWidthTabHandle (page 2703) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewColorComplementUPP (page 2713) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewColorSearchUPP (page 2713) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewDeviceLoopDrawingUPP (page 2713) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewDragGrayRgnUPP (page 2714) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewGWorldFromPtr (page 2718) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDArcUPP (page 2721) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDBitsUPP (page 2721) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDCommentUPP (page 2721) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDGetPicUPP (page 2722) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDJShieldCursorUPP (page 2722) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDLineUPP (page 2722) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDOpcodeUPP (page 2723) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDOvalUPP (page 2723) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDPolyUPP (page 2723) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDPutPicUPP (page 2724) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDRectUPP (page 2724) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDRgnUPP (page 2724) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDRRectUPP (page 2725) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDStdGlyphsUPP (page 2725) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDTextUPP (page 2725) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewQDTxMeasUPP (page 2726) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

NewRegionToRectsUPP (page 2726) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

OffscreenVersion (page 2730) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

OpenCursorComponent (page 2735) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDAddRectToDirtyRegion (page 2755) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDAddRegionToDirtyRegion (page 2756) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDDisplayWaitCursor (page 2757) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDDisposeRegionBits (page 2757) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGetCursorData (page 2761) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.5
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGetDirtyRegion (page 2762) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGetPictureBounds (page 2762) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGlobalToLocalPoint (page 2763) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGlobalToLocalRect (page 2763) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDGlobalToLocalRegion (page 2764) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDIsNamedPixMapCursorRegistered (page 2764) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDIsPortBufferDirty (page 2764) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDIsPortBuffered (page 2765) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDLocalToGlobalPoint (page 2765) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDLocalToGlobalRect (page 2765) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDLocalToGlobalRegion (page 2766) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDRegisterNamedPixMapCursor (page 2770) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDRestoreRegionBits (page 2771) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSaveRegionBits (page 2771) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSetCursorScale (page 2772) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSetDirtyRegion (page 2772) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSetNamedPixMapCursor (page 2772) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSetPatternOrigin (page 2773) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSwapPort (page 2773) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSwapPortTextFlags (page 2773) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDSwapTextFlags (page 2774) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

QDUnregisterNamedPixMapCursor (page 2774) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SectRegionWithPortClipRegion (page 2786) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SectRegionWithPortVisibleRegion (page 2786) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetCursorComponent (page 2793) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortBackPixPat (page 2800) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortBounds (page 2801) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortClipRegion (page 2801) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortCustomXFerProc (page 2802) Deprecated in Mac OS X v10.4

SetPortFillPixPat (page 2802) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortFracHPenLocation (page 2803) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortGrafProcs (page 2803) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortOpColor (page 2803) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortPenMode (page 2804) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortPenPixPat (page 2804) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortPenSize (page 2805) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextFace (page 2806) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextFont (page 2806) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextMode (page 2806) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortTextSize (page 2807) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetPortVisibleRegion (page 2807) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetQDError (page 2808) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetQDGlobalsArrow (page 2808) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

SetQDGlobalsRandomSeed (page 2809) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

StdOpcode (page 2819) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`StuffHex` (page 2823) Deprecated in Mac OS X v10.4
Sets byte values into memory. (Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SwapPortPicSaveHandle` (page 2824) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SwapPortPolySaveHandle` (page 2825) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

`SwapPortRegionSaveHandle` (page 2825) Deprecated in Mac OS X v10.4
(Deprecated. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Functions

AddComp

Adds a function to the head of the current device data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void AddComp (
   ColorComplementUPP compProc
);
```

Parameters

compProc
A pointer to your complement function, `ColorComplementProcPtr` (page 2834).

Discussion

`AddComp` creates and allocates a `CProcRec` (page 2852) data structure.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

AddPt

Adds the coordinates of two points.

```
void AddPt (
   Point src,
   Point *dst
);
```

Parameters

src
A point, the coordinates of which are to be added to the point in the `dstPt` parameter.

dst

A pointer to a point, the coordinates of which are to be added to the point in the `srcPt` parameter. On return, this value contains the result of adding the coordinates of the points you supplied in the `srcPt` and `dstPt` parameters.

Discussion
The `AddPt` function adds the coordinates of the point specified in the `srcPt` parameter to the coordinates of the point specified in the `dstPt` parameter, and returns the result in the `dstPt` parameter.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`QuickdrawAPI.h`

AddSearch

Adds a function to the head of the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void AddSearch (
   ColorSearchUPP searchProc
);
```

Parameters

searchProc

A pointer to your custom search function, `ColorSearchProcPtr` (page 2834).

Discussion
`AddSearch` creates and allocates an `SProcRec` (page 2883) data structure.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

AllocCursor

Reallocates cursor memory. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void AllocCursor (
   void
);
```

Discussion
Under normal circumstances, you should never need to use this function, since Color QuickDraw handles reallocation of cursor memory.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

AllowPurgePixels

Makes the base address for an offscreen pixel image purgeable. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void AllowPurgePixels (
   PixMapHandle pm
);
```

Parameters

pm
A handle to an offscreen pixel map.

Discussion
The `AllowPurgePixels` function allows the Memory Manager to free the memory it occupies if available memory space becomes low. By default, `NewGWorld` creates an unpurgeable base address for an offscreen pixel image.

To get a handle to an offscreen pixel map, first use the `GetGWorldPixMap` (page 2642) function. Then supply this handle for the `pm` parameter of `AllowPurgePixels`.

Your application should call the `LockPixels` (page 2704) function before drawing into or copying from an offscreen pixel map. If the Memory Manager has purged the base address for its pixel image, `LockPixels` returns `FALSE`. In that case either your application should use the `UpdateGWorld` (page 2831) function to begin reconstructing the offscreen pixel image, or it should draw directly to an onscreen graphics port.

Only unlocked memory blocks can be made purgeable. If you use `LockPixels`, you must use the `UnlockPixels` function before calling `AllowPurgePixels`.

Special Considerations

The `AllowPurgePixels` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

AngleFromSlope

Converts a slope value to an angle value.

```
short AngleFromSlope (
   Fixed slope
);
```

Parameters

slope

The slope, defined as Dx/Dy, which is the horizontal change divided by the vertical change between any two points on a line with the slope.

Return Value
The angle corresponding to the slope specified in the `slope` parameter treated `MOD` 180. Angles are defined in clockwise degrees from 12 o'clock. The negative y-axis is defined as being at 12 o'clock, and the positive y-axis at 6 o'clock. The x-axis is defined as usual, with the positive side defined as being at 3 o'clock.

Special Considerations

The `AngleFromSlope` function is most useful when you require speed more than accuracy in performing the calculation. The integer result is within 1 degree of the correct answer, but not necessarily within half a degree.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

BackColor

Changes a basic graphics port's background color. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void BackColor (
   long color
);
```

Parameters

color

One of eight color values. See "Color Constants" (page 2885).

Discussion
The background color is the color of the pixels in the bitmap wherever no drawing has taken place. By default the background color of a `GrafPort` is white.

The `BackColor` function sets the background color for the current graphics port to the color that you specify in the `color` parameter. When you draw with the `patCopy` and `srcCopy` transfer modes, for example, white pixels are drawn in the color you specify with `BackColor`.

All nonwhite colors appear as black on black-and-white screens. Before you use `BackColor`, use the `DeviceLoop` function to determine the color characteristics of the current screen.

Special Considerations

The `BackColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Version Notes
In System 7, use the Color QuickDraw function `RGBBackColor`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

BackPat

Changes the bit pattern used as the background pattern by the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void BackPat (
   const Pattern *pat
);
```

Parameters

pat

A bit pattern, as defined by a `Pattern` (page 2866) structure.

Discussion
The `BackPat` function sets the bit pattern defined in the `Pattern` structure, which you specify in the `pat` parameter, to be the background pattern. (The standard bit patterns `white`, `black`, `gray`, `ltGray`, and `dkGray` are predefined; the initial background pattern for the graphics port is `white`.) This pattern is stored in the `bkPat` field of a `GrafPort` structure.

The `BackPat` function also sets a bit pattern for the background color in a color graphics port. The `BackPat` function creates a handle, of type `PixPatHandle`, for the bit pattern and stores this handle in the `bkPixPat` field of the `CGrafPort` structure. As in basic graphics ports, Color QuickDraw draws patterns in color graphics ports at the time of drawing, not at the time you use `PenPat` to set the pattern.

To define your own patterns, you typically create pattern, 'PAT', or pattern list, 'PAT#', resources.

Special Considerations

The `BackPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

BackPixPat

Assigns a pixel pattern as the background pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void BackPixPat (
   PixPatHandle pp
);
```

Parameters

pp
A handle to the pixel pattern to use as the background pattern.

Discussion
Setting the background pattern allows the ScrollRect function and the shape-erasing functions (for example, EraseRect) to fill the background with a colored patterned "ink."

The BackPixPat function is similar to the basic QuickDraw function BackPat, except that you pass BackPixPat a handle to a multicolored pixel pattern instead of a bit pattern.

The handle to the pixel pattern is stored in the bkPixPat field of the CGrafPort structure, therefore, you should not dispose of this handle since QuickDraw removes all references to your pattern from an existing graphics port when you dispose of it.

If you use BackPixPat to set a background pixel pattern in a basic graphics port, the data in the pat1Data field of the PixPat (page 2871) structure is placed into the bkPat field of the GrafPort structure.

To define your own pixel pattern, create a pixel pattern resource, x is described on 'ppat', or use the NewPixPat (page 2720) function. To set the background pattern to a bit pattern, you can also use the QuickDraw function, BackPat.

Special Considerations

The BackPixPat function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

BitMapToRegion

Converts a bitmap or pixel map to a region. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr BitMapToRegion (
   RgnHandle region,
   const BitMap *bMap
);
```

Parameters

region

A handle to a region to hold the converted `BitMap` or `PixMap` structure. This must be a valid region handle created with the `NewRgn` function. The old region contents are lost.

bMap

A pointer to a `BitMap` or `PixMap` structure to be converted. If you supply a `PixMap` structure, its pixel depth must be 1.

Return Value
A result code.

Discussion
The `BitMapToRegion` function converts a given `BitMap` or `PixMap` structure to a region. Pixels are added to the region where the corresponding entries in the bitmap have a value of 1. You would generally use this region later for drawing operations.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CalcCMask

Determines where filling will not occur when filling from the outside of a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CalcCMask (
   const BitMap *srcBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *dstRect,
   const RGBColor *seedRGB,
   ColorSearchUPP matchProc,
   long matchData
);
```

Parameters

srcBits

The source image. If the image is in a pixel map, you must coerce its `PixMap` structure to a `BitMap` structure.

dstBits

The destination image. The `CalcCMask` function returns the generated bitmap mask in this parameter. You can then use this mask with the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions.

srcRect

The rectangle of the source image.

dstRect
The rectangle of the destination image.

seedRGB
An `RGBColor` structure specifying the color for pixels that should not be filled.

matchProc
An optional matching function.

matchData
Data for the optional matching function.

Discussion
Specify a source image in the `srcBits` parameter and in the `srcRect` parameter, specify a rectangle within that source image. Starting from the edges of this rectangle, `CalcCMask` calculates which pixels cannot be filled. By default, `CalcCMask` returns 1's in the mask to indicate which pixels have the exact color that you specify in the `seedRGB` parameter and which pixels are enclosed by shapes whose outlines consist entirely of pixels with this color.

For instance, if the source image in `srcBits` contains a dark blue rectangle on a red background, and your application sets `seedRGB` equal to dark blue, then `CalcCMask` returns a mask with 1's in the positions corresponding to the edges and interior of the rectangle, and the 0's outside of the rectangle.

If you set the `matchProc` and `matchData` parameters to 0, `CalcCMask` uses the exact color specified in the `RGBColor` structure that you supply in the `seedRGB` parameter. You can customize `CalcCMask` by writing your own color search function and pointing to it in the `matchProc` parameter. As with `SeedCFill`, you can then use the `matchData` parameter in any manner useful for your application.

The `CalcCMask` function does not scale so the source and destination rectangles must be the same size. Calls to `CalcCMask` are not clipped to the current port and are not stored into QuickDraw pictures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CalcMask

Determines where filling will not occur when filling from the outside of a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CalcMask (
   const void *srcPtr,
   void *dstPtr,
   short srcRow,
   short dstRow,
   short height,
   short words
);
```

Parameters

srcPtr
A pointer to the source bit image.

dstPtr
A pointer to the destination bit image.

srcRow
Row width of the source bitmap.

dstRow
Row width of the destination bitmap.

height
Height (in pixels) of the fill rectangle.

words
Width (in words) of the fill rectangle.

Discussion
The `CalcMask` function produces a bit image with 1's in all pixels to which paint could not flow from any of the outer edges of the rectangle. Use this bit image as a mask with the `CopyBits` or `CopyMask` function. A hollow object produces a solid mask, but an open object produces a mask of itself.

As with the `SeedFill` function, point to the bit image you want to fill with the `srcPtr` parameter, which can point to the image's base address or a word boundary within the image. Specify a pixel height and word width with the `height` and `words` parameters to define a fill rectangle that delimits the area you want to fill. The fill rectangle can be the entire bit image or a subset of it. Point to a destination image with the `dstPtr` parameter. Specify the row widths of the source and destination bitmaps (their `rowBytes` values) with the `srcRow` and `dstRow` parameters. (The bitmaps can be different sizes, but they must be large enough to contain the fill rectangle at the origins specified by `srcPtr` and `dstPtr`.)

Calls to `CalcMask` are not clipped to the current port and are not stored into QuickDraw pictures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ClipCGContextToRegion

Sets the clipping path in a Quartz 2D graphics context, using a clipping region. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus ClipCGContextToRegion (
   CGContextRef gc,
   const Rect *portRect,
   RgnHandle region
);
```

Parameters
context
A Quartz context associated with a graphics port. You can obtain such a context by calling `QDBeginCGContext` (page 2756).

portRect
The `portRect` for the graphics port associated with the context.

region
A region that represents the desired clipping path.

Return Value
A result code. If `noErr`, the clipping path is now the region-based path.

Discussion
This function sets the clipping path in the specified context to closely approximate the geometry of the specified region.

Unlike clipping in Quartz 2D, this function does not intersect the new region-based path with the current clipping path—the new path simply replaces the current clipping path.

You should use this function only when absolutely necessary—it's relatively inefficient when compared to Quartz 2D clipping functions such as `CGContextClipToRect`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ClipRect

Changes the clipping region of the current graphics port (basic or color). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ClipRect (
   const Rect *r
);
```

Parameters
r
A pointer to a rectangle for the boundary of the new clipping region. The `ClipRect` function changes the clipping region of the current graphics port to a region that's equivalent to this rectangle. `ClipRect` doesn't change the region handle, but it affects the clipping region itself.

Discussion
Since `ClipRect` makes a copy of the given rectangle, any subsequent changes you make to that rectangle do not affect the clipping region of the port.

The `ClipRect` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CloseCursorComponent

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr CloseCursorComponent (
   ComponentInstance ci
);
```

Return Value
A result code.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ClosePicture

Completes the collection of drawing commands and picture comments that define your picture. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ClosePicture (
   void
);
```

Discussion
The `ClosePicture` function stops collecting drawing commands and picture comments for the currently open picture. You should perform one and only one call to `ClosePicture` for every call to the `OpenCPicture` (or `OpenPicture`) function.

The `ClosePicture` function calls the `ShowPen` function, balancing the call made by `OpenCPicture` (or `OpenPicture`) to the `HidePen` function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ClosePoly

Completes the collection of lines that defines a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ClosePoly (
   void
);
```

Discussion
The `ClosePoly` function stops collecting line-drawing commands for the currently open polygon and computes the `polyBBox` field of the `Polygon` (page 2873) structure. You should call `ClosePoly` only once for every call to the `OpenPoly` function.

The `ClosePoly` function uses the `ShowPen` function, balancing the call to the `HidePen` function made by the `OpenPoly` function.

Special Considerations

The `ClosePoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CloseRgn

Organizes a collection of lines and shapes into a region definition. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CloseRgn (
   RgnHandle dstRgn
);
```

Parameters

`dstRgn`

The handle to the region to close. This handle should be a region handle returned by the `NewRgn` (page 2726) function.

Discussion
The `CloseRgn` function stops the collection of lines and framed shapes, organizes them into a region definition, and saves the result in the region whose handle you pass in the `dstRgn` parameter.

The `CloseRgn` function does not create the destination region; you must have already allocated space for it by using the `OpenRgn` function. The `CloseRgn` function calls the `ShowPen` function, balancing the call to the `HidePen` function made by `OpenRgn`.

When you no longer need the memory occupied by the region, use the `DisposeRgn` (page 2609) function.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

Regions are limited to 32 KB in size in basic QuickDraw and 64 KB in Color QuickDraw. When you structure drawing operations in an open region, the resulting region description may overflow this limit. Should this happen in Color QuickDraw, the `QDError` function returns the result code `regionTooBigError`. Since the resulting region is potentially corrupt, the `CloseRgn` function returns an empty region if it detects `QDError` has returned `regionTooBigError`.

The `CloseRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

Color2Index

Obtains the index of the best available approximation for a given color in the color table of the current `GDevice` data structure. This function is used only by system software. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
long Color2Index (
   const RGBColor *myColor
);
```

Parameters

myColor

A pointer to the RGB color value to be approximated.

Return Value

The index of the best approximation for the given color that is available in the color table of the current `GDevice` data structure. Note that `Color2Index` returns a long integer, in which the low-order word is the index value; the high-order word contains zeros.

Discussion

You should not call `Color2Index` from within a custom search function (described in `ColorSearchProcPtr` (page 2834)).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

ColorBit

Sets the foreground color for all printing in the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ColorBit (
   short whichBit
);
```

Parameters

whichBit

An integer specifying the plane to draw into.

Discussion

The `ColorBit` function is called by printing software for a color printer (or other color-imaging software) to set the `GrafPort` structure's `colorBit` field to the value in the `whichBit` parameter. This value tells QuickDraw which plane of the color picture to draw into. QuickDraw draws into the plane corresponding to the bit number specified by the `whichBit` parameter. Since QuickDraw can support output devices that have up to 32 bits of color information per pixel, the possible range of values for `whichBit` is 0 through 31. The initial value of the `colorBit` field is 0.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

CopyBits

Copies a portion of a bitmap or a pixel map from one graphics port or offscreen graphics world into another graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CopyBits (
   const BitMap *srcBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short mode,
   RgnHandle maskRgn
);
```

Parameters

srcBits

The source `BitMap` structure.

dstBits

The destination `BitMap` structure.

srcRect

The source rectangle.

dstRect

The destination rectangle.

mode

One of the eight source modes in which the copy is to be performed. See "Source, Pattern, and Arithmetic Transfer Mode Constants" (page 2898). The `CopyBits` function always dithers images when shrinking them between pixel maps on direct devices.

When transferring pixels from a source pixel map to a destination pixel map, color QuickDraw interprets the source mode constants differently than basic QuickDraw does.

When you use `CopyBits` on a computer running color QuickDraw, you can also specify one of the transfer modes in the `mode` parameter.

maskRgn

A region to use as a clipping mask. You can pass a region handle to specify a mask region the resulting image is always clipped to this mask region and to the boundary rectangle of the destination bitmap. If the destination bitmap is the current graphics port's bitmap, it is also clipped to the intersection of the graphics port's clipping region and visible region. If you do not want to clip to a masking region, just pass `NULL` for this parameter.

Discussion

The `CopyBits` function transfers any portion of a bitmap between two basic graphics ports, or any portion of a pixel map between two color graphics ports. Use `CopyBits` to move offscreen graphic images into an onscreen window, to blend colors for the image in a pixel map, and to shrink and expand images.

Specify a source bitmap in the `srcBits` parameter and a destination bitmap in the `dstBits` parameter. When copying images between color graphics ports, you must coerce each `CGrafPort` structure to a `GrafPort` structure, dereference the `portBits` fields of each, and then pass these "bitmaps" in the `srcBits` and `dstBits` parameters. If your application copies a pixel image from a color graphics port called `MyColorPort`, for example, you could specify `(* GrafPtr(MyColorPort)).portBits` in the `srcBits` parameter. In a `CGrafPort` structure, the high 2 bits of the `portVersion` field are set. This field, which shares the same position in a `CGrafPort` structure as the `portBits.rowBytes` field in a `GrafPort` structure, indicates to `CopyBits` that you have passed it a handle to a pixel map rather than a bitmap.

Using the `srcRect` and `dstRect` parameters, you can specify identically or differently sized source and destination rectangles; for differently sized rectangles, `CopyBits` scales the source image to fit the destination. If the bit image is a circle in a square source rectangle, and the destination rectangle is not square, the bit image appears as an oval in the destination. When you specify rectangles in the `srcRect` and `dstRect` parameters, use the local coordinate systems of, respectively, the source and destination graphics ports.

The `CopyDeepMask` (page 2586) function combines the functions of the `CopyBits` and `CopyMask` functions.

Special Considerations

When you use the `CopyBits` function to transfer an image between pixel maps, the source and destination images may be of different pixel depths, of different sizes, and they may have different color tables. However, `CopyBits` assumes that the destination pixel map uses the same color table as the color table for the current `GDevice` structure. (This is because the Color Manager requires an inverse table for translating the color table from the source pixel map to the destination pixel map.)

The `CopyBits` function applies the foreground and background colors of the current graphics port to the image in the destination pixel map (or bitmap), even if the source image is a bitmap. This causes the foreground color to replace all black pixels in the destination and the background color to replace all white pixels. To avoid unwanted coloring of the image, use the `RGBForeColor` function to set the foreground to black and use the `RGBBackColor` function to set the background to white before calling `CopyBits`.

The source bitmap or pixel map must not occupy more memory than half the available stack space. The stack space required by `CopyBits` is roughly five times the value of the `rowBytes` field of the source pixel map: one `rowBytes` value for the pixel map (or bitmap), an additional `rowBytes` value for dithering, another

rowBytes value when stretching or shrinking the source pixel map into the destination, another rowBytes value for any color map changing, and a fifth additional rowBytes value for any color aliasing. If there is insufficient memory to complete a CopyBits operation in Color QuickDraw, the QDError function returns the result code –143.

If you use CopyBits to copy between two graphics ports that overlap, you must first use the LocalToGlobal function to convert to global coordinates, and then specify the global variable screenBits for both the srcBits and dstBits parameters.

The CopyBits function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

If you are reading directly from a NuBus video card with a base address of Fs00000 and there is not a card in the slot (s–1) below it, CopyBits reads addresses less than the base address of the pixel map. This causes a bus error. To work around the problem, remap the baseAddr field of the pixel map in your video card to at least 20 bytes above the NuBus boundary; an address link of Fs000020 precludes the problem.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

CopyDeepMask

Uses a mask when copying bitmaps or pixel maps between graphics ports (or from an offscreen graphics world into a graphics port). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CopyDeepMask (
   const BitMap *srcBits,
   const BitMap *maskBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *maskRect,
   const Rect *dstRect,
   short mode,
   RgnHandle maskRgn
);
```

Parameters

srcBits
The source BitMap structure.

maskBits
The masking BitMap structure.

dstBits
The destination BitMap structure. The result is clipped to the mask region that you specify in the maskRgn parameter, and to the boundary rectangle that you specify in the dstRect parameter.

srcRect
The source rectangle.

maskRect

The mask rectangle. This must be the same size as the rectangle passed in the `srcRect` parameter. The rectangle you pass here selects the portion of the bitmap or pixel map that you specify in the `maskBits` parameter to use as the mask.

dstRect

The destination rectangle.

mode

The source mode.

maskRgn

The mask clipping region. If you do not want to clip to the mask region, specify `NULL`.

Discussion

`CopyDeepMask` combines the effects of the `CopyBits` and `CopyMask` functions. You specify a mask to `CopyDeepMask` so that it transfers the source image to the destination image only where the bits of the mask are set to 1. Use `CopyDeepMask` to move offscreen graphic images into an onscreen window, to blend colors for the image in a pixel map, and to shrink and expand images.

When copying images between color graphics ports, you must coerce each port's `CGrafPort` structure to a `GrafPort` structure, dereference the `portBits` fields of each, and then pass these "bitmaps" in the `srcBits` and `dstBits` parameters. If your application copies a pixel image from a color graphics port called `MyColorPort`, for example, you could specify (* `GrafPtr(MyColorPort)).portBits` in the `srcBits` parameter. The transfer can be performed in any of the transfer modes—with or without adding the `ditherCopy` constant—that are available to `CopyBits` (page 2584).

Using the `srcRect` and `dstRect` parameters, you can specify identically or differently sized source and destination rectangles; for differently sized rectangles, `CopyDeepMask` scales the source image to fit the destination. When you specify rectangles in the `srcRect` and `dstRect` parameters, use the local coordinate systems of, respectively, the source and destination graphics ports.

If you specify pixel maps to `CopyDeepMask`, they may range from 1 to 32 pixels in depth. The pixel depth of the mask that you specify in the `maskBits` parameter is applied as a filter between the source and destination pixel maps that you specify in the `srcBits` and `dstBits` parameters. A black mask pixel value means that the copy operation is to take the source pixel a white value means that the copy operation is to take the destination pixel. Intermediate values specify a weighted average, which is calculated on a color component basis. For each pixel's color component value, the calculation is

(1 – mask) x source + (mask) x destination

Thus high mask values for a pixel's color component reduce that component's contribution from the source `PixMap` structure.

Special Considerations

As with the `CopyMask` function, calls to `CopyDeepMask` are not recorded in pictures and do not print.

See the list of special considerations for `CopyBits` (page 2584); these considerations also apply to `CopyDeepMask`.

The `CopyDeepMask` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CopyMask

Copies a bit or pixel image from one graphics port or offscreen graphics world into another graphics port only where the bits in a mask are set to 1. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CopyMask (
   const BitMap *srcBits,
   const BitMap *maskBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *maskRect,
   const Rect *dstRect
);
```

Parameters

srcBits
The source `BitMap` structure.

maskBits
The mask `BitMap` structure.

dstBits
The destination `BitMap` structure.

srcRect
The source rectangle.

maskRect
The mask rectangle. This must be the same size as the rectangle passed in the `srcRect` parameter. The rectangle you pass in this parameter selects the portion of the bitmap or pixel map that you specify in the `maskBits` parameter to use as the mask.

dstRect
The destination rectangle.

Discussion
The `CopyMask` function copies the source bitmap or pixel map that you specify in the `srcBits` parameter to a destination bitmap or pixel map that you specify in the `dstBits` parameter—but only where the bits of the mask bitmap or pixel map that you specify in the `maskBits` parameter are set to 1. When copying images between color graphics ports, you must coerce each `CGrafPort` structure to a `GrafPort` structure, dereference the `portBits` fields of each, and then pass these "bitmaps" in the `srcBits` and `dstBits` parameters. If your application copies a pixel image from a color graphics port called `MyColorPort`, for example, you could specify (* `GrafPtr(MyColorPort)).portBits` in the `srcBits` parameter.

Using the `srcRect` and `dstRect` parameters, you can specify identically or differently sized source and destination rectangles; for differently sized rectangles, `CopyMask` scales the source image to fit the destination. When you specify rectangles in the `srcRect` and `dstRect` parameters, use the local coordinate systems of, respectively, the source and destination graphics ports.

If you specify pixel maps to `CopyMask`, they may range from 1 to 32 pixels in depth. The pixel depth of the mask that you specify in the `maskBits` parameter is applied as a filter between the source and destination pixel maps that you specify in the `srcBits` and `dstBits` parameters. A black mask pixel value means that the copy operation is to take the source pixel a white value means that the copy operation is to take the destination pixel. Intermediate values specify a weighted average, which is calculated on a color component basis. For each pixel's color component value, the calculation is

(1 – mask) x source + (mask) x destination

Thus high mask values for a pixel's color component reduce that component's contribution from the source `PixMap` structure.

Use the bitmap returned by `CalcMask` (page 2578) as the mask in order to implement a mask copy similar to that performed by the MacPaint lasso tool. In the same way, you can use the pixel map returned by the `CalcCMask` function.

The `CopyDeepMask` (page 2586) function combines the functions of the `CopyMask` and `CopyBits` functions.

Special Considerations

Calls to `CopyMask` are not recorded in pictures and do not print.

See the list of special considerations for `CopyBits` (page 2584); these considerations also apply to `CopyMask`.

The `CopyMask` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CopyPixMap

Duplicates a `PixMap` structure. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CopyPixMap (
   PixMapHandle srcPM,
   PixMapHandle dstPM
);
```

Parameters

srcPM
A handle to the `PixMap` structure to be copied.

dstPM
On return, a handle to the duplicated `PixMap` structure.

Discussion
Typically, you do not need to call this function in your application code, because the `CopyPixMap` function copies the contents of the source `PixMap` structure to the destination `PixMap` structure. The contents of the color table are copied, so the destination `PixMap` has its own copy of the color table. Because the `baseAddr` field of the `PixMap` structure is a pointer, the pointer, but not the image itself, is copied.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CopyPixPat

Copies the contents of one pixel pattern to another. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CopyPixPat (
   PixPatHandle srcPP,
   PixPatHandle dstPP
);
```

Parameters

srcPP
A handle to a source pixel pattern, the contents of which you want to copy.

dstPP
A handle to a destination pixel pattern, into which you want to copy the contents of the pixel pattern in the `srcPP` parameter.

Discussion
The `CopyPixPat` function copies all of the fields in the source `PixPat` (page 2871) structure, including the contents of the data handle, expanded data handle, expanded map, pixel map handle, and color table.

Generally, your application should create a pixel pattern in a `'ppat'` resource, instead of using this function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CopyRgn

Makes a copy of a region.

```
void CopyRgn (
    RgnHandle srcRgn,
    RgnHandle dstRgn
);
```

Parameters

srcRgn

A handle to the region to copy.

dstRgn

A handle to the region to receive the copy.

Discussion

The `CopyRgn` function copies the mathematical structure of the region whose handle you pass in the `srcRgn` parameter into the region whose handle you pass in the `dstRgn` parameter; that is, `CopyRgn` makes a duplicate copy of `srcRgn`. When calling `CopyRgn`, pass handles that have been returned by the `NewRgn` function in the `srcRgn` and `dstRgn` parameters.

Once this is done, the region indicated by `srcRgn` may be altered (or even disposed of) without affecting the region indicated by `dstRgn`. The `CopyRgn` function does not create the destination region; space must already have been allocated for it by using the `NewRgn` function.

Special Considerations

The `CopyRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

CreateCGContextForPort

Creates a Quartz 2D drawing environment associated with a graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

Not Recommended

```
OSStatus CreateCGContextForPort (
   CGrafPtr inPort,
   CGContextRef *outContext
);
```

Parameters

port

A color graphics port in which to draw. Offscreen graphics worlds with pixel depths of 1, 2, 4, and 8 are not supported. When using Quartz 2D to draw in a offscreen graphics world, alpha information is always ignored. Printing ports are not supported—if you specify a printing port, this function does nothing and returns a non-zero result code.

contextPtr

A pointer to your storage for a Quartz context. Upon completion, `contextPtr` points to a context associated with the port. The context matches the port's pixel depth, width, and height. Otherwise the context is in a default state and does not necessarily match other port attributes such as foreground color, background color, or clip region.

You should release this context when you no longer need it.

Return Value
A result code. If `noErr`, the context was successfully created.

Discussion
This function is not recommended in Mac OS X version 10.1 and later. For information about its replacement, see `QDBeginCGContext` (page 2756).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CreateNewPort

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CGrafPtr CreateNewPort (
   void
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

CreateNewPortForCGDisplayID

Creates a graphics port associated with a display. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CGrafPtr CreateNewPortForCGDisplayID (
   UInt32 inCGDisplayID
);
```

Parameters

displayID

A display identifier. If the identifier is not valid, the main display is used instead. For information about finding displays, see *Quartz Display Services Reference*.

Return Value

A new display port. The `portBounds` rectangle is the same size as the display. When you are finished using the port, you should call `DisposePort` (page 2603) to release it.

Discussion

This function returns a graphics port used to draw directly to a display. The pixel map for the new port is taken from the `GDevice` record corresponding to the display. There is no back buffer associated with the port.

Before calling this function, you should capture the display. For information about capturing displays, see *Quartz Display Services Reference*.

You should not call this function and then attempt to create a Quartz drawing environment inside the port. Instead, applications using Quartz 2D can call `CGDisplayGetDrawingContext` (page 1497) to obtain a context suitable for drawing directly to a captured display.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

CTabChanged

Signals QuickDraw that the content of a `ColorTable` structure has been modified. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void CTabChanged (
   CTabHandle ctab
);
```

Parameters

ctab

A handle to the `ColorTable` structure changed by your application.

Discussion

The `CTabChanged` function calls the function `GetCTSeed` and gets a new, unique identifier in the `ctSeed` field of the `ColorTable` structure, and notifies QuickDraw of the change.

Your application should never need to directly modify a `ColorTable` structure and use the `CTabChanged` function; instead, your application should use the QuickDraw functions provided for manipulating the values in a `ColorTable` structure.

Special Considerations

The `CTabChanged` function may move or purge memory in the application heap; do not call the `CTabChanged` function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In

`QDOffscreen.h`

CursorComponentChanged

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr CursorComponentChanged (
   ComponentInstance ci
);
```

Return Value

A result code.

Carbon Porting Notes

This function is not implemented on Mac OS X.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

CursorComponentSetData

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr CursorComponentSetData (
   ComponentInstance ci,
   long data
);
```

Return Value

A result code.

Carbon Porting Notes

This function is not implemented on Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

DelComp

Removes a custom complement function from the current `GDevice` data structure's list of complement functions. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DelComp (
   ColorComplementUPP compProc
);
```

Parameters

compProc
A pointer to the complement function, `ColorComplementProcPtr` (page 2834), to be deleted. `DelComp` disposes of the chain element, but does nothing to the `ProcPtr` data structure.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

DelSearch

Removes a custom search function from the current `GDevice` data structure's list of search functions. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DelSearch (
   ColorSearchUPP searchProc
);
```

Parameters

searchProc
A pointer to the custom search function, `ColorSearchProcPtr` (page 2834) to be deleted. `DelSearch` disposes of the chain element, but does nothing to the `ProcPtr` data structure.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

DeltaPoint

Subtracts the coordinates of one point from another. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
long DeltaPoint (
   Point ptA,
   Point ptB
);
```

Parameters

p1

The first point.

p2

The second point, the coordinates of which are to be subtracted from the coordinates of the first point.

Return Value

A 32-bit value that contains the differences between the coordinates of the points `p1` and `p2`. The vertical difference is returned in the high 16 bits and the horizontal difference is returned in the low 16 bits.

Discussion

You should not cast the result to a `Point` data structure. Instead, use `HiWord` and `LoWord` to obtain the horizontal and vertical differences.

For example:

```
Point pointDiff;
SInt32 difference = DeltaPoint(p1, p2);
pointDiff.h = LoWord(difference);
pointDiff.v = HiWord(difference);
```

While `DeltaPoint` is supported in Carbon, you can achieve the same result in a more direct manner using the function `SubPt` (page 2824).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

deltapoint

(Deprecated in Mac OS X v10.4. Use `DeltaPoint` (page 2596) instead.)

```
long deltapoint (
   Point *ptA,
   Point *ptB
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

DeviceLoop

Draws images that are optimized for every screen they cross. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DeviceLoop (
   RgnHandle drawingRgn,
   DeviceLoopDrawingUPP drawingProc,
   long userData,
   DeviceLoopFlags flags
);
```

Parameters

drawingRgn
A handle to the region in which you will draw; this drawing region uses coordinates that are local to its graphics port.

drawingProc
A pointer to your own drawing function.

userData
Any additional data that you wish to supply to your drawing function.

flags
One or more members of the set of flags defined by the " Device Loop Flags" (page 2889) data type. if you want to use the default behavior of DeviceLoop, specify an empty set ([]) in this parameter.

Discussion
The DeviceLoop function searches for graphics devices that intersect your window's drawing region, and it calls your drawing function for each dissimilar video device it finds.

Because DeviceLoop provides your drawing function with the pixel depth and other attributes of each video device, your drawing function can optimize its drawing for each video device.

See DeviceLoopDrawingProcPtr (page 2836) for a description of the drawing function you must provide for the drawingProc parameter.

Special Considerations

The DeviceLoop function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

DiffRgn

Subtracts one region from another.

```
void DiffRgn (
   RgnHandle srcRgnA,
   RgnHandle srcRgnB,
   RgnHandle dstRgn
);
```

Parameters

srcRgnA

A handle to the region to subtract from.

srcRgnB

A handle to the region to subtract.

dstRgn

On return, a handle to the region holding the resulting area. If the first source region is empty, `DiffRgn` sets the destination to the empty region defined by the rectangle (0,0,0,0).

The `DiffRgn` function does not create the destination region; you must have already allocated memory for it by using the `NewRgn` (page 2726) function.

The destination region may be one of the source regions, if desired.

Discussion

The DiffRgn procedure subtracts the region whose handle you pass in the srcRgnB parameter from the region whose handle you pass in the srcRgnA parameter and places the difference in the region whose handle you pass in the dstRgn parameter. If the first source region is empty, DiffRgn sets the destination to the empty region defined by the rectangle (0,0,0,0).

The DiffRgn procedure does not create the destination region; you must have already allocated memory for it by using the NewRgn function. The destination region may be one of the source regions, if desired.

Special Considerations

The `DiffRgn` function may temporarily use heap space that's twice the size of the two input regions.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

DisposeCCursor

Disposes of all structures allocated by the `GetCCursor` function. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeCCursor (
   CCrsrHandle cCrsr
);
```

Parameters

cCrsr

A handle to the color cursor to be disposed of.

Discussion
Use `DisposeCCursor` for each call to the `GetCCursor` (page 2635) function.

The `DisposeCCursor` function is also available as the `DisposCCursor` function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

DisposeColorComplementUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeColorComplementUPP (
   ColorComplementUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeColorSearchUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeColorSearchUPP (
   ColorSearchUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeCTable

Disposes a `ColorTable` structure. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeCTable (
   CTabHandle cTable
);
```

Parameters

cTable

A handle to a `ColorTable` structure to dispose of.

Discussion

The DisposeCTable procedure disposes of the ColorTable record whose handle you pass in the cTable parameter.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

DisposeDeviceLoopDrawingUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeDeviceLoopDrawingUPP (
   DeviceLoopDrawingUPP userUPP
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`QuickdrawTypes.h`

DisposeDragGrayRgnUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeDragGrayRgnUPP (
   DragGrayRgnUPP userUPP
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`QuickdrawTypes.h`

DisposeGDevice

Disposes of a `GDevice` structure, releases the space allocated for it, and disposes of all the data structures allocated for it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeGDevice (
   GDHandle gdh
);
```

Parameters

gdh

A handle to the `GDevice` structure.

Discussion

Generally, you should never need to use this function. Color QuickDraw calls this function when appropriate. The `DisposeGDevice` function is also available as the `DisposGDevice` function.

When your application uses the `DisposeGWorld` function to dispose of an offscreen graphics world, `DisposeGDevice` disposes of its `GDevice` structure.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

DisposeGWorld

Disposes of all the memory allocated for an offscreen graphics world. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeGWorld (
   GWorldPtr offscreenGWorld
);
```

Parameters

offscreenGWorld

A pointer to an offscreen graphics world. In this parameter, pass the pointer returned to your application by the `NewGWorld` function when you created the offscreen graphics world.

Discussion

The `DisposeGWorld` function disposes of all the memory allocated for the specified offscreen graphics world, including the pixel map, color table, pixel image, and `GDevice` structure (if one was created).

Call `DisposeGWorld` only when your application no longer needs the pixel image associated with this offscreen graphics world. If this offscreen graphics world was the current device, the current device is reset to the device stored in the global variable `MainDevice`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
QTCarbonShell
WhackedTV

Declared In
`QDOffscreen.h`

DisposePixMap

Disposes a `PixMap` structure and its color table. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposePixMap (
   PixMapHandle pm
);
```

Parameters

pm

A handle to the `PixMap` structure to be disposed of.

Discussion
The `CloseCPort` function calls `DisposePixMap`.

Your application typically does not need to call this function. This function is also available as `DisposPixMap`.

If your application uses `DisposePixMap`, take care that it does not dispose of a `PixMap` structure whose color table is the same as the current device's CLUT.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
LiveVideoMixer2

Declared In
`QuickdrawAPI.h`

DisposePixPat

Releases the storage allocated to a pixel pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposePixPat (
   PixPatHandle pp
);
```

Parameters

pp

A handle to the pixel pattern to be disposed of.

Discussion
The `DisposePixPat` function disposes of the data handle, expanded data handle, and pixel map handle allocated to the pixel pattern that you specify in the `ppat` parameter.

The `DisposePixPat` function is also available as the `DisposPixPat` function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

DisposePort

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposePort (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

DisposeQDArcUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDArcUPP (
   QDArcUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDBitsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDBitsUPP (
   QDBitsUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDCommentUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDCommentUPP (
   QDCommentUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDGetPicUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDGetPicUPP (
   QDGetPicUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDJShieldCursorUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDJShieldCursorUPP (
   QDJShieldCursorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDLineUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDLineUPP (
   QDLineUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDOpcodeUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDOpcodeUPP (
   QDOpcodeUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDOvalUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDOvalUPP (
   QDOvalUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

DisposeQDPolyUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDPolyUPP (
   QDPolyUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

DisposeQDPutPicUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDPutPicUPP (
   QDPutPicUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

DisposeQDRectUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDRectUPP (
   QDRectUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDRgnUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDRgnUPP (
   QDRgnUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDRRectUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDRRectUPP (
   QDRRectUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDStdGlyphsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDStdGlyphsUPP (
   QDStdGlyphsUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDTextUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDTextUPP (
   QDTextUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeQDTxMeasUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeQDTxMeasUPP (
   QDTxMeasUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

DisposeRegionToRectsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeRegionToRectsUPP (
   RegionToRectsUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawAPI.h`

DisposeRgn

Releases the memory occupied by a region.

```
void DisposeRgn (
   RgnHandle rgn
);
```

Parameters

rgn

A handle to the region to dispose. This handle should be a region handle returned by the `NewRgn` (page 2726) function.

Discussion
Use `DisposeRgn` only after you are completely through with a region.

Special Considerations

The `DisposeRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`QuickdrawAPI.h`

DisposeScreenBuffer

Disposes an offscreen graphics world. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DisposeScreenBuffer (
   PixMapHandle offscreenPixMap
);
```

Parameters

offscreenPixMap

A handle to an existing offscreen `PixMap` structure.

Discussion
Generally, applications do not need to use `DisposeScreenBuffer`. The `DisposeGWorld` (page 2601) function uses the `DisposeScreenBuffer` function when disposing of an offscreen graphics world.

The `DisposeScreenBuffer` function disposes of the memory allocated for the base address of an offscreen pixel image.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

DrawPicture

Draws a picture on any type of output device. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void DrawPicture (
   PicHandle myPicture,
   const Rect *dstRect
);
```

Parameters

myPicture

A handle to the picture to be drawn. You must access a picture through its handle.

When creating pictures, the `OpenCPicture` (page 2734) and `OpenPicture` (page 2736) functions return their handles. You can use the `GetPicture` (page 2648) function to get a handle to a QuickDraw picture stored in a `'PICT'` resource. To get a handle to a QuickDraw picture stored in a `'PICT'` file, you must use File Manager functions. To get a picture stored in the scrap, use the Scrap Manager function `GetScrap` to get a handle to its data and then coerce this handle to one of type `PicHandle`.

dstRect

A destination rectangle, specified in coordinates local to the current graphics port, in which to draw the picture. The `DrawPicture` function shrinks or expands the picture as necessary to align the borders of its bounding rectangle with the rectangle you specify in this parameter. To display a picture at a resolution other than that at which it was created, your application should compute an appropriate destination rectangle by scaling its width and height by the following factor:

```
scale factor = destination resolution / source resolution
```

For example, if a picture was created at 300 dpi and you want to display it at 75 dpi, then your application should compute the destination rectangle width and height as 1/4 of those of the picture's bounding rectangle. Use the `GetPictInfo` function to gather information about a picture. The `PictInfo` structure returned by `GetPictInfo` returns the picture's resolution in its `hRes` and `vRes` fields. The `sourceRect` field contains the bounding rectangle for displaying the image at its optimal resolution.

Discussion
Within the rectangle that you specify in the `dstRect` parameter, the `DrawPicture` function draws the picture that you specify in the `myPicture` parameter.

The DrawPicture function passes any picture comments to the StdComment function pointed to by the commentProc field of the CQDProcs or QDProcs structure, which in turn is pointed to by the grafProcs field of a CGrafPort or GrafPort structure. The default StdComment function provided by QuickDraw does no comment processing whatsoever. If you want to process picture comments when drawing a picture, use the SetStdCProcs function to assist you in changing the CQDProcs structure and use the SetStdProcs function to assist you in changing the QDProcs structure.

Special Considerations

Always use the ClipRect function to specify a clipping region appropriate for your picture before defining it with the OpenCPicture (or OpenPicture) function. If you do not use ClipRect to specify a clipping region, OpenCPicture uses the clipping region specified in the current graphics port. If the clipping region is very large (as it is when a graphics port is initialized) and you want to scale the picture, the clipping region can become invalid when DrawPicture scales the clipping region—in which case, your picture will not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when DrawPicture draws it. Setting a clipping region equal to the port rectangle of the current graphics port always sets a valid clipping region.

When it scales, DrawPicture changes the size of the font instead of scaling the bits. However, the widths used by bitmap fonts are not always linear. For example, the 12-point width isn't exactly 1/2 of the 24-point width. This can cause lines of text to become slightly longer or shorter as the picture is scaled. The difference is often insignificant, but if you are trying to draw a line of text that fits exactly into a box (a spreadsheet cell, for example), the difference can become noticeable to the user—most typically, at print time. The easiest way to avoid such problems is to specify a destination rectangle that is the same size as the bounding rectangle for the picture. Otherwise, your application may need to directly process the opcodes in the picture instead of using DrawPicture.

You may also have disappointing results if the fonts contained in an image are not available on the user's system. Before displaying a picture, your application may want to use the Picture Utilities to determine what fonts are contained in the picture, and then use Font Manager functions to determine whether the fonts are available on the user's system. If they are not, you can use Dialog Manager functions to display an alert box warning the user of display problems.

If there is insufficient memory to draw a picture in Color QuickDraw, the QDError function returns the result code noMemForPictPlaybackErr.

The DrawPicture function may move or purge memory.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

QuickdrawAPI.h

EmptyRect

Determines whether a rectangle is an empty rectangle.

```
Boolean EmptyRect (
   const Rect *r
);
```

Parameters

r

The rectangle to examine.

Return Value

`TRUE` if the rectangle that you specify in the `r` parameter is an empty rectangle, `FALSE` if it is not. A rectangle is considered empty if the bottom coordinate is less than or equal to the top coordinate or if the right coordinate is less than or equal to the left.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

SoftVDigX

Declared In

`QuickdrawAPI.h`

EmptyRgn

Determines whether a region is empty.

```
Boolean EmptyRgn (
   RgnHandle rgn
);
```

Parameters

rgn

A handle to the region to test for emptiness.

Return Value

`TRUE` if the region whose handle you pass in the `rgn` parameter is an empty region or `FALSE` if it is not.

Discussion

The `EmptyRgn` function does not create an empty region. To create an empty region, you can perform any of the following operations:

- Use `NewRgn` (page 2726).
- Pass the handle to an empty region to `CopyRgn` (page 2590).
- Pass an empty rectangle to either `SetRectRgn` (page 2810) or `RectRgn` (page 2776).

- Call `CloseRgn` (page 2582) without a previous call to `OpenRgn` (page 2737).
- Call `CloseRgn` (page 2582) without performing any drawing after calling `OpenRgn` (page 2737).
- Pass an empty region to `OffsetRgn` (page 2732).
- Pass an empty region or too large an inset to `InsetRgn` (page 2673)
- Pass two nonintersecting regions to `SectRgn` (page 2786).
- Pass two empty regions to `UnionRgn` (page 2828).
- Pass two identical or nonintersecting regions to `DiffRgn` (page 2598) or `XorRgn` (page 2833).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

EqualPt

Determines whether the coordinates of two given points are equal.

```
Boolean EqualPt (
   Point pt1,
   Point pt2
);
```

Parameters

pt1
The first of two points to be compared.

pt2
The second of two points to be compared.

Return Value
`TRUE` if the coordinates of the two points are equal, or `FALSE` if they are not.

Discussion
The EqualPt function compares the points specified in the pt1 and pt2 parameters and returns TRUE if their coordinates are equal or FALSE if they are not.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawAPI.h`

EqualRect

Determines whether two rectangles are equal.

```
Boolean EqualRect (
    const Rect * rect1,
    const Rect * rect2
);
```

Parameters

rect1

The first of two rectangles to compare.

rect2

The second of two rectangles to compare.

Return Value

`TRUE` if the rectangles are equal, `FALSE` if they are not.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawAPI.h`

EqualRgn

Determines whether two regions have identical sizes, shapes, and locations.

```
Boolean EqualRgn (
    RgnHandle rgnA,
    RgnHandle rgnB
);
```

Parameters

rgnA

A handle to the first of two regions to compare.

rgnB

A handle to the second of two regions to compare.

Return Value

`TRUE` if the two regions are equal; `FALSE` if they are not. The two regions must have identical sizes, shapes, and locations to be considered equal. Any two empty regions are always equal.

Discussion

The EqualRgn function compares the two regions whose handles you pass in the rgnA and rgnB parameters and returns TRUE if they're equal or FALSE if they're not.

The two regions must have identical sizes, shapes, and locations to be considered equal. Any two empty regions are always equal.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

EraseArc

Erases a wedge. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void EraseArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

r
The rectangle that defines an oval's boundaries.

startAngle
The angle indicating the start of the arc.

arcAngle
The angle indicating the arc's extent.

Discussion
Using the patCopy pattern mode, the EraseArc function draws a wedge of the oval bounded by the rectangle that you specify in the r parameter with the background pattern for the current graphics port. As in FrameArc (page 2629) , use the startAngle and arcAngle parameters to define the arc of the wedge.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The EraseArc function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

EraseOval

Erases an oval. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void EraseOval (
   const Rect *r
);
```

Parameters

r

The rectangle that defines the oval's boundary.

Discussion

Using the background pattern for the current graphics port and the `patCopy` pattern mode, the `EraseOval` function draws the interior of an oval just inside the bounding rectangle that you specify in the `r` parameter. This effectively erases the oval bounded by the specified rectangle.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `EraseOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

ErasePoly

Erases a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ErasePoly (
   PolyHandle poly
);
```

Parameters

poly

A handle to the polygon to erase. The `OpenPoly` (page 2737) function returns this handle when you first create the polygon.

Discussion

Using the `patCopy` pattern mode, the `ErasePoly` function draws the interior of the polygon whose handle you pass in the `poly` parameter with the background pattern for the current graphics port.

This function leaves the location of the graphics pen unchanged.

This function temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 2707) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Special Considerations

The `ErasePoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

EraseRect

Erases a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void EraseRect (
   const Rect *r
);
```

Parameters

r

The rectangle to erase.

Discussion
Using the `patCopy` pattern mode, the `EraseRect` function draws the interior of the rectangle that you specify in the `r` parameter with the background pattern for the current graphics port. This effectively erases the rectangle, making the shape blend into the background pattern of the graphics port. For example, use `EraseRect` to erase the port rectangle for a window before redrawing into the window.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `EraseRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
HID Explorer

Declared In
`QuickdrawAPI.h`

EraseRgn

Erases a region. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void EraseRgn (
   RgnHandle rgn
);
```

Parameters

rgn

The region to erase.

Discussion

Using the `patCopy` pattern mode, the `EraseRgn` function draws the interior of the region whose handle you pass in the `rgn` parameter with the background pattern for the current graphics port.

This function leaves the location of the graphics pen unchanged.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Special Considerations

The `EraseRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

EraseRoundRect

Erases a rounded rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void EraseRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

r

The rectangle that defines the rounded rectangle's boundaries.

ovalWidth

The width of the oval defining the rounded corner.

ovalHeight
The height of the oval defining the rounded corner.

Discussion
Using the `patCopy` pattern mode, the `EraseRoundRect` function draws the interior of the rounded rectangle bounded by the rectangle that you specify in the `r` parameter with the background pattern of the current graphics port. This effectively erases the rounded rectangle. Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners of the rounded rectangle.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `EraseRoundRect` function may move or purge memory blocks in the application; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FillArc

Fills a wedge with any available bit pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillArc (
   const Rect *r,
   short startAngle,
   short arcAngle,
   const Pattern *pat
);
```

Parameters

r
The rectangle that defines an oval's boundaries.

startAngle
The angle indicating the start of the arc.

arcAngle
The bit pattern to use for the fill.

pat
The angle indicating the arc's extent.

Discussion
Using the `patCopy` pattern mode and the pattern defined in the `Pattern` (page 2866) structure that you specify in the `pat` parameter, the `FillArc` function draws a wedge of the oval bounded by the rectangle that you specify in the `r` parameter. As in `FrameArc` (page 2629) use the `startAngle` and `arcAngle` parameters to define the arc of the wedge.

This function leaves the location of the graphics pen unchanged.

Use `GetPattern` (page 2646) and `GetIndPattern` (page 2643) to get a pattern stored in a resource.

Use `PaintArc` (page 2740) to draw a wedge with the pen pattern for the current graphics port.

To fill a wedge with a pixel pattern, use the `FillCArc` function.

Special Considerations

The `FillArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FillCArc

Fills a wedge with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillCArc (
   const Rect *r,
   short startAngle,
   short arcAngle,
   PixPatHandle pp
);
```

Parameters

r
The rectangle that defines the oval's boundaries.

startAngle
The angle indicating the start of the arc.

arcAngle
The angle indicating the arc's extent.

pp
A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion
Use the `startAngle` and `arcAngle` parameters to define the arc of the wedge. This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FillCOval

Fills an oval with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillCOval (
   const Rect *r,
   PixPatHandle pp
);
```

Parameters

r

The rectangle containing the oval to be filled.

pp

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion
This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FillCPoly

Fills a polygon with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillCPoly (
   PolyHandle poly,
   PixPatHandle pp
);
```

Parameters

poly

A handle to the polygon to be filled.

pp

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion
This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FillCRect

Fills a rectangle with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillCRect (
   const Rect *r,
   PixPatHandle pp
);
```

Parameters

r
The rectangle to be filled.

pp
A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion
This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FillCRgn

Fills a region with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillCRgn (
   RgnHandle rgn,
   PixPatHandle pp
);
```

Parameters

rgn

A handle to the region to be filled.

pp

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion

This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

FillCRoundRect

Fills a rounded rectangle with the given pixel pattern, using the `patCopy` pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillCRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight,
   PixPatHandle pp
);
```

Parameters

r

The rectangle that defines the rounded rectangle's boundaries.

ovalWidth

The width of the oval defining the rounded corner.

ovalHeight

The height of the oval defining the rounded corner.

pp

A handle to the `PixPat` structure for the pixel pattern to be used for the fill.

Discussion

Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners. This function ignores the `pnPat`, `pnMode`, and `bkPat` fields of the current graphics port and leaves the pen location unchanged.

Special Considerations

The `FillCRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

FillOval

Fills an oval with any available bit pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillOval (
   const Rect *r,
   const Pattern *pat
);
```

Parameters

r

The rectangle that defines the oval's boundaries.

pat

The bit pattern to use for the fill.

Discussion

Using the `patCopy` pattern mode and the bit pattern defined in the `Pattern` (page 2866) structure that you specify in the `pat` parameter, the `FillOval` function draws the interior of an oval just inside the bounding rectangle that you specify in the `r` parameter. The pen location does not change.

Use `GetPattern` (page 2646) and `GetIndPattern` (page 2643) , to get a pattern stored in a resource. Use the `PaintOval` (page 2741) function to draw the interior of an oval with the pen pattern for the current graphics port.

To fill an oval with a pixel pattern, use the `FillCOval` function.

Special Considerations

The `FillOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

FillPoly

Fills a polygon with any available bit pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillPoly (
   PolyHandle poly,
   const Pattern *pat
);
```

Parameters

poly

A handle to the polygon to fill. The `OpenPoly` (page 2737) function returns this handle when you first create the polygon.

pat

The bit pattern to use for the fill.

Discussion

Using the `patCopy` pattern mode, the `FillPoly` function draws the interior of the polygon whose handle you pass in the `poly` parameter with the pattern defined in the `Pattern` (page 2866) structure that you specify in the `pat` parameter.

This function leaves the location of the graphics pen unchanged.

This function temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 2707) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Use `GetPattern` (page 2646) and `GetIndPattern` (page 2643) to get a pattern stored in a resource.

Use `PaintPoly` (page 2741) to draw the interior of a polygon with the pen pattern for the current graphics port. To fill a polygon with a pixel pattern, use the `FillCPoly` function.

Special Considerations

The `FillPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

FillRect

Fills a rectangle with any available bit pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillRect (
    const Rect * r,
    const Pattern * pat
);
```

Parameters

r

The rectangle to fill.

pat

The bit pattern to use for the fill.

Discussion

Using the `patCopy` pattern mode, the `FillRect` function draws the interior of the rectangle that you specify in the `r` parameter with the pattern defined in the `Pattern` (page 2866) structure that you specify in the `pat` parameter. This function leaves the pen location unchanged.

Use `GetPattern` (page 2646) and `GetIndPattern` (page 2643) , to get a pattern stored in a resource.

Use the `PaintRect` (page 2742) to draw the interior of a rectangle with the pen pattern for the current graphics port. To fill a rectangle with a pixel pattern, use the `FillCRect` function.

Special Considerations

The `FillRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

FillRgn

Fills a region with any available bit pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillRgn (
    RgnHandle rgn,
    const Pattern * pat
);
```

Parameters

rgn

A handle to the region to fill.

pat

The bit pattern to use for the fill.

Discussion
Using the `patCopy` pattern mode, the `FillRgn` function draws the interior of the region with the pattern defined in the `Pattern` (page 2866) structure that you specify in the `pat` parameter.

This function leaves the location of the graphics pen unchanged.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Use `GetPattern` (page 2646) and `GetIndPattern` (page 2643) to get a pattern stored in a resource.

Use `PaintRgn` (page 2743) to draw the interior of a region with the pen pattern for the current graphics port. To fill a region with a pixel pattern, use the `FillCRegion` function.

Special Considerations

The `FillRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FillRoundRect

Fills a rounded rectangle with any available bit pattern. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FillRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight,
   const Pattern *pat
);
```

Parameters

r
The rectangle that defines the rounded rectangle's boundaries.

ovalWidth
The width of the oval defining the rounded corner.

ovalHeight
The height of the oval defining the rounded corner.

pat
The bit pattern to use for the fill.

Discussion
Using the `patCopy` pattern mode, the `FillRoundRect` function draws the interior of the rounded rectangle bounded by the rectangle that you specify in the `r` parameter with the bit pattern defined in the `Pattern` structure that you specify in the `pat` parameter. Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners. The pen location does not change.

To fill a rounded rectangle with a pixel pattern, use the `FillCRoundRect` function.

Use `GetPattern` (page 2646) and `GetIndPattern` (page 2643) to get a pattern stored in a resource. Use `PaintRoundRect` (page 2743) to draw the interior of a rounded rectangle with the pen pattern for the current graphics port.

Special Considerations

The `FillRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ForeColor

Changes the color of the "ink" used for framing, painting, and filling on computers that support only basic QuickDraw. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ForeColor (
   long color
);
```

Parameters

color

One of eight color values. See "Color Constants" (page 2885).

Discussion
By default, the foreground color of a `GrafPort` is black.

The `ForeColor` function sets the foreground color for the current graphics port to the color that you specify in the `color` parameter. When you draw with the `patCopy` and `srcCopy` transfer modes, for example, black pixels are drawn in the color you specify with `ForeColor`.

When printing, use the `ColorBit` (page 2584) function to set the foreground color.

All nonwhite colors appear as black on black-and-white screens. Before you use `ForeColor`, use the `DeviceLoop` function to determine the color characteristics of the current screen.

Special Considerations

The `ForeColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Version Notes
In System 7, you may instead use the color QuickDraw function `RGBForeColor`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar

Declared In
`QuickdrawAPI.h`

FrameArc

Draws an arc of the oval that fits inside a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FrameArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

r
The rectangle that defines an oval's boundaries.

startAngle
The angle indicating the start of the arc.

arcAngle
The angle indicating the arc's extent.

Discussion
Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameArc` function draws an arc of the oval bounded by the rectangle that you specify in the `r` parameter. Use the `startAngle` parameter to specify where the arc begins as modulo 360. Use the `arcAngle` parameter to specify how many degrees the arc covers. Specify whether the angles are in positive or negative degrees a positive angle goes clockwise, while a negative angle goes counterclockwise. Zero degrees is at 12 o'clock high, 90 (or –270) is at 3 o'clock, 180 (or –180) is at 6 o'clock, and 270 (or –90) is at 9 o'clock. Measure other angles relative to the bounding rectangle.

A line from the center of the rectangle through its upper-right corner is at 45, even if the rectangle is not square a line through the lower-right corner is at 135, and so on.

The arc is as wide as the pen width and as tall as the pen height. The pen location does not change.

Special Considerations

The `FrameArc` function differs from other QuickDraw functions that frame shapes in that the arc is not mathematically added to the boundary of a region that's open and being formed.

The `FrameArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FrameOval

Draws an outline inside an oval. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FrameOval (
   const Rect *r
);
```

Parameters

r

The rectangle that defines the oval's boundary.

Discussion
Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameOval` function draws an outline just inside the oval with the bounding rectangle that you specify in the `r` parameter. The outline is as wide as the pen width and as tall as the pen height. The pen location does not change.

If a region is open and being formed, the outside outline of the new oval is mathematically added to the region's boundary.

Special Considerations

The `FrameOval` function may move or purge memory blocks in the application; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FramePoly

Draws the outline of a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FramePoly (
   PolyHandle poly
);
```

Parameters

poly

A handle to the polygon to draw. The `OpenPoly` (page 2737) function returns this handle when you first create the polygon.

Discussion

Using the current graphics port's pen pattern, pattern mode, and size, the `FramePoly` function plays back the line-drawing commands that define the polygon whose handle you pass in the `poly` parameter.

The graphics pen hangs below and to the right of each point on the boundary of the polygon. Thus, the drawn polygon extends beyond the right and bottom edges of the polygon's bounding rectangle (which is stored in the `polyBBox` field of the `Polygon` structure) by the pen width and pen height, respectively. All other graphics operations, such as painting a polygon with the `PaintPoly` function, occur strictly within the boundary of the polygon.

If a polygon is open and being formed, `FramePoly` affects the outline of the polygon just as if the line-drawing functions themselves had been called. If a region is open and being formed, the outside outline of the polygon being framed is mathematically added to the region's boundary.

The result of this function is undefined whenever any horizontal or vertical line through the polygon would intersect the polygon's outline more than 50 times.

Special Considerations

The `FramePoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

FrameRect

Draws an outline inside a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FrameRect (
    const Rect * r
);
```

Parameters

r

The rectangle to frame.

Discussion

Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameRect` function draws an outline just inside the rectangle that you specify in the `r` parameter. The outline is as wide as the pen width and as tall as the pen height. The pen location does not change.

If a region is open and being formed, the outside outline of the new rectangle is mathematically added to the region's boundary.

Special Considerations

The `FrameRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FrameRgn

Draws an outline inside a region. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FrameRgn (
    RgnHandle rgn
);
```

Parameters

rgn
A handle to the region to frame.

Discussion
Using the current graphics port's pen pattern, pattern mode, and pen size, the `FrameRgn` function draws an outline just inside the region whose handle you pass in the rgn parameter. The outline never goes outside the region boundary. The pen location does not change.

If a region is open and being formed, the outside outline of the region being framed is mathematically added to that region's boundary.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined. The `FrameRgn` function in particular requires that there would be no more than 25 such intersections.

Special Considerations

The `FrameRgn` function calls the functions `CopyRgn`, `InsetRgn`, and `DiffRgn`, so `FrameRgn` may temporarily use heap space that's three times the size of the original region.

The `FrameRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

FrameRoundRect

Draws an outline inside a rounded rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void FrameRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

r
The rectangle that defines the rounded rectangle's boundaries.

ovalWidth
The width of the oval defining the rounded corner.

ovalHeight
The height of the oval defining the rounded corner.

Discussion
Using the pattern, pattern mode, and size of the graphics pen for the current graphics port, the `FrameRoundRect` function draws an outline just inside the rounded rectangle bounded by the rectangle that you specify in the `r` parameter. The outline is as wide as the pen width and as tall as the pen height. The pen location does not change.

Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners of the rounded rectangle.

If a region is open and being formed, the outside outline of the new rounded rectangle is mathematically added to the region's boundary.

Special Considerations

The `FrameRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GDeviceChanged

Notifies QuickDraw that the content of a `GDevice` structure has been modified. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GDeviceChanged (
   GDHandle gdh
);
```

Discussion
If your application changes the `pmTable` field of the `PixMap` structure specified in a `GDevice` structure, call `GDeviceChanged`. If your application changes the content of the `ColorTable` structure referenced by the `PixMap` structure, call both `GDeviceChanged` and `CTabChanged`.

Your application should never need to directly modify a `GDevice` structure and use the `GDeviceChanged` function; instead, your application should use the QuickDraw functions described in this book for manipulating the values in a `GDevice` structure.

Special Considerations

The `GDeviceChanged` function may move or purge memory in the application heap; do not call the `GDeviceChanged` function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

GetBackColor

Obtains the background color of the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetBackColor (
   RGBColor *color
);
```

Parameters

color

On return, the `RGBColor` structure for the current background color.

Discussion
This function operates for graphics ports defined by both the `GrafPort` and `CGrafPort` structures. If the current graphics port is defined by a `CGrafPort` structure, the returned value is taken directly from the `rgbBkColor` field.

If the current graphics port is defined by a `GrafPort` structure, then only eight possible colors can be returned. These eight colors are determined by the values in a global variable named `QDColors`, which is a handle to a color table containing the current QuickDraw colors.

Use the `RGBBackColor` (page 2779) function to change the background color.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetCCursor

Loads a color cursor resource into memory. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CCrsrHandle GetCCursor (
   short crsrID
);
```

Parameters

crsrID

The resource ID of the cursor that you want to display.

Return Value
A handle to the new `CCrsr` structure. To display this cursor on the screen, call `SetCCursor`. If a resource with the specified ID isn't found, then this function returns a `NULL` handle.

Discussion
The `GetCCursor` function creates a new `CCrsr` (page 2847) structure and initializes it using the information in the 'crsr' resource with the specified ID.

Since the `GetCCursor` function creates a new `CCrsr` structure each time it is called, do not call the `GetCCursor` function before each call to the `SetCCursor` function. Unlike the way `GetCursor` and `SetCursor` are normally used, `GetCCursor` does not dispose of or detach the resource, so resources of type `'crsr'` should typically be purgeable. Call the `DisposeCCursor` (page 2598) function when you are finished using the color cursor created with `GetCCursor`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetClip

Saves the clipping region of the current graphics port (basic or color). (**Deprecated.** Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetClip (
   RgnHandle rgn
);
```

Parameters

rgn

A handle to the region to be clipped. The `GetClip` function changes this region to one that's equivalent to the clipping region of the current graphics port. The `GetClip` function doesn't change the region handle.

Discussion
You can use the `GetClip` and `SetClip` functions to preserve the current clipping region: use `GetClip` to save the current port's clipping region, and use `SetClip` to restore it. If, for example, you want to draw a half-circle on the screen, you can set the clipping region to half of the square that would enclose the whole circle, and then draw the whole circle. Only the half within the clipping region is actually drawn in the graphics port.

The `GetClip` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.k.h`

GetCPixel

Determines the color of an individual pixel specified in the `h` and `v` parameters. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetCPixel (
   short h,
   short v,
   RGBColor *cPix
);
```

Parameters

h
The horizontal coordinate of the point at the upper-left corner of the pixel.

v
The vertical coordinate of the point at the upper-left corner of the pixel.

cPix
On return, the `RGBColor` structure for the pixel color.

Discussion
Use the `SetCPixel` (page 2791) function to change the color of this pixel.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetCTable

Obtains a color table stored in a `'clut'` resource. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CTabHandle GetCTable (
   short ctID
);
```

Parameters

ctID

The resource ID of a `'clut'` resource.

Return Value

A handle to the color table. If the `'clut'` resource with that ID is not found, `GetCTable` returns `NULL`. Before you place this handle in the `pmTable` field of a `PixMap` structure, first use the `DisposeCTable` function to dispose of the handle already there.

Discussion

Before you modify a `ColorTable` structure, change its `ctSeed` field to invalidate it. To do this, use the `CTabChanged` (page 2593) function.

The `GetCTable` function recognizes a number of standard `'clut'` resource IDs. You can obtain the default grayscale color table for a given pixel depth by calling `GetCTable`, adding 32 (decimal) to the pixel depth, and passing these values in the `ctID` parameter:

- A pixel depth of 1. Pass a resource ID of 33. Color table composition: black, white.
- A pixel depth of 2. Pass a resource ID of 34. Color table composition: black, 33% gray, 66% gray, white.
- A pixel depth of 4. Pass a resource ID of 36. Color table composition: black, 14 shades of gray, white.
- A pixel depth of 8. Pass a resource ID of 40. Color table composition: black, 254 shades of gray, white.

For full color, obtain the default color tables by adding 64 to the pixel depth and passing these values in the `ctID` parameter:

- A pixel depth of 2. Pass a resource ID of 66. Color table composition: black, 50% gray, highlight color, white.
- A pixel depth of 4. Pass a resource ID of 68. Color table composition: black, 14 colors including the highlight color, white.
- A pixel depth of 8. Pass a resource ID of 72. Color table composition: black, 254 colors including the highlight color, white.

Special Considerations

The `GetCTable` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetCTSeed

Obtains a unique seed value for a color table created by your application. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
long GetCTSeed (
   void
);
```

Return Value

A unique seed value that you can use in the `ctSeed` field of a color table created by your application. It is greater than the value stored in the constant `minSeed`.

Discussion

The seed value guarantees that the color table is recognized as distinct from the destination, and that color table translation is performed properly.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetCursor

Loads a cursor resource into memory. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CursHandle GetCursor (
    short cursorID
);
```

Parameters

cursorID

The resource ID for the cursor you want to display. You can supply one of the "Cursor ID Constants" (page 2886) to get a handle to one of the standard cursors.

Return Value

A handle to a `Cursor` structure for the cursor with the resource ID that you specify in the `cursorID` parameter. If the resource cannot be read into memory, `GetCursor` returns `NULL`.

Discussion

To get a handle to a color cursor, use the `GetCCursor` (page 2635) function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetDeviceList

Obtains a handle to the first `GDevice` structure in the device list. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle GetDeviceList (
   void
);
```

Return Value
A handle to the first `GDevice` structure in the global variable `DeviceList`.

Discussion
All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations

The `GetDeviceList` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetForeColor

Obtains the color of the foreground color for the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetForeColor (
   RGBColor *color
);
```

Parameters

color

On return, the `RGBColor` structure for the current foreground color.

Discussion
This function operates for graphics ports defined by both the `GrafPort` and `CGrafPort` structures. If the current graphics port is defined by a `CGrafPort` structure, the returned value is taken directly from the `rgbFgColor` field.

If the current graphics port is defined by a `GrafPort` structure, then only eight possible RGB values can be returned. These eight values are determined by the values in a global variable named `QDColors`, which is a handle to a color table containing the current QuickDraw colors.

Use the `RGBForeColor` (page 2780) function to change the foreground color.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`QuickdrawAPI.h`

GetGDevice

Obtains a handle to the `GDevice` structure for the current device. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle GetGDevice (
   void
);
```

Return Value
A handle to the current device.

Discussion
At any given time, exactly one video device is the current device—that is, the one on which drawing is actually taking place.

Color QuickDraw stores a handle to the current device in the global variable `TheGDevice`.

All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

You can also use the `GetGWorld` function to get a handle to the `GDevice` structure for the current device.

Special Considerations

The `GetGDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetGWorld

Saves the current graphics port (basic, color, or offscreen) and the current `GDevice` structure. (**Deprecated.** Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetGWorld (
   CGrafPtr *port,
   GDHandle *gdh
);
```

Parameters

port

On return, a pointer to the current graphics port in the `port` parameter. This parameter can return values of type `GrafPtr`, `CGrafPtr`, or `GWorldPtr`, depending on whether the current graphics port is a basic graphics port, color graphics port, or offscreen graphics world.

gdh

On return, a pointer to a handle to the `GDevice` structure for the current device.

Discussion

After using `GetGWorld` to save a graphics port and a `GDevice` structure, use the `SetGWorld` (page 2796) function to restore them.

Special Considerations

The `GetGWorld` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

Simple DrawSprocket

Declared In

`ImageCompression.k.h`

GetGWorldDevice

Obtains a handle to the `GDevice` structure associated with an offscreen graphics world. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle GetGWorldDevice (
   GWorldPtr offscreenGWorld
);
```

Parameters

offscreenGWorld

A pointer to an offscreen graphics world. The pointer returned to your application by the `NewGWorld` function.

Return Value

A handle to the `GDevice` structure associated with the offscreen graphics world specified by the `offscreenGWorld` parameter.

If you created the offscreen world by specifying the `noNewDevice` flag, the `GDevice` structure is for one of the screen devices or is the `GDevice` structure that you specified to `NewGWorld` or `UpdateGWorld`.

If you point to a `GrafPort` or `CGrafPort` structure in the `offscreenGWorld` parameter, `GetGWorldDevice` returns the current device.

Special Considerations

The `GetGWorldDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

WhackedTV

Declared In

`QDOffscreen.h`

GetGWorldPixMap

Obtains the pixel map created for an offscreen graphics world. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixMapHandle GetGWorldPixMap (
   GWorldPtr offscreenGWorld
);
```

Parameters

`offscreenGWorld`

A pointer to an offscreen graphics world. Pass the pointer returned to your application by the `NewGWorld` (page 2715) function when you created the offscreen graphics world.

Return Value

A handle to the pixel map created for an offscreen graphics world. Your application can, in turn, pass the handle returned by `GetGWorldPixMap` as a parameter to other QuickDraw functions that accept a handle to a pixel map.

On a system running only basic QuickDraw, the `GetGWorldPixMap` function returns the handle to a 1-bit pixel map that your application can supply as a parameter to the other functions related to offscreen graphics worlds. However, your application should not supply this handle to color QuickDraw functions.

Special Considerations

To ensure compatibility on systems running basic QuickDraw instead of Color QuickDraw, use `GetGWorldPixMap` whenever you need to gain access to the bitmap created for a graphics world—that is, do not dereference the `GWorldPtr` structure for that graphics world.

Version Notes

The `GetGWorldPixMap` function is not available in systems preceding System 7. You can make sure that the `GetGWorldPixMap` function is available by using the `Gestalt` function with the `gestaltSystemVersion` selector. Test the low-order word in the `response` parameter; if the value is $0700 or greater, then `GetGWorldPixMap` is available.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
ASCIIMoviePlayerSample
QTCarbonShell
WhackedTV

Declared In
QDOffscreen.h

GetIndPattern

Obtains a pattern stored in a pattern list (`'PAT#'`) resource. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetIndPattern (
   Pattern *thePat,
   short patternListID,
   short index
);
```

Parameters

thePat
On return, a pointer to a `Pattern` (page 2866) structure for the pattern stored in the specified pattern list resource.

patternListID
The resource ID for a resource of type `'PAT#'`.

index
The index number for the desired pattern within the pattern list (`'PAT#'`) resource. The index number can range from 1 to the number of patterns in the pattern list resource.

Discussion
The `GetIndPattern` function calls the following Resource Manager function with these parameters:

```
GetResource('PAT#', patternListID);
```

There is a pattern list resource in the System file that contains the standard Macintosh patterns used by MacPaint. The resource ID is represented by the constant `sysPatListID`.

Special Considerations

The `GetIndPattern` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

GetMainDevice

Obtains a handle to the `GDevice` structure for the main screen. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle GetMainDevice (
   void
);
```

Return Value
A handle to the device for the main screen, which is the device containing the menu bar.

Discussion
A handle to the main device is kept in the global variable `MainDevice`.

All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations

The `GetMainDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetMaskTable

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Ptr GetMaskTable (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetMaxDevice

Obtains a handle to the `GDevice` structure for the video device with the greatest pixel depth. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle GetMaxDevice (
   const Rect *globalRect
);
```

Parameters

globalRect

A rectangle, in global coordinates, that intersects the graphics devices that you are searching to find the one with the greatest pixel depth.

Return Value
A handle to the device with the greatest pixel depth.

Discussion
All existing `GDevice` structures are linked together in the device list. After using this function to obtain a handle to the current `GDevice` structure, your application can use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations

The `GetMaxDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetNextDevice

Returns a handle to the next `GDevice` structure in the device list. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle GetNextDevice (
   GDHandle curDevice
);
```

Parameters

curDevice

A handle to the `GDevice` structure at which you want the search to begin.

Return Value
A handle to the next device. If there are no more `GDevice` structures in the list, `NULL`.

Discussion
After using any of the functions `GetDeviceList` (page 2639) , `GetGDevice` (page 2640) , `GetMainDevice` (page 2644) , or `GetMaxDevice` (page 2644) to obtain a handle to a `GDevice` structure, use the `GetNextDevice` function to obtain a handle to the next `GDevice` structure in the list.

Special Considerations

The `GetNextDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPattern

Obtains a pattern (`'PAT'`) resource stored in a resource file. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PatHandle GetPattern (
   short patternID
);
```

Parameters

`patternID`
The resource ID for a resource of type 'PAT'.

Return Value
a handle to the pattern having the resource ID that you specify in the `patID` parameter. If a pattern resource with the ID that you request does not exist, the `GetPattern` function returns `NULL`.

Discussion
The `GetPattern` function calls the following Resource Manager function with these parameters:

GetResource('PAT', patID);

When you are finished using the pattern, dispose of its handle with the Memory Manager function `DisposeHandle`.

Special Considerations

The `GetPattern` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPen

Determines the location of the graphics pen. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetPen (
   Point *pt
);
```

Parameters

pt

On return, a pointer to the graphics pen's current position in the current graphics port. The point returned is in the local coordinates of the current graphics port.

Discussion
In the pt parameter, the GetPen procedure returns the current pen position. The point returned is in the local coordinates of the current graphics port.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPenState

Determines the graphics pen's location, size, pattern, and pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetPenState (
   PenState *pnState
);
```

Parameters

pnState

On return, a pointer to a `PenState` structure holding information about the graphics pen. The `GetPenState` function saves the location, size, pattern, and pattern mode of the graphics pen for the current graphics port in this structure.

Discussion
After changing the graphics pen as necessary, restore these pen states with the `SetPenState` (page 2798) function.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPicture

Obtains a handle to a picture stored in a `'PICT'` resource. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PicHandle GetPicture (
   short pictureID
);
```

Parameters

pictureID

The resource ID for a `'PICT'` resource.

Return Value

A handle to the picture in the specified 'PICT' resource. To draw the picture stored in the resource, pass this handle to the `DrawPicture` (page 2610) function. If the resource cannot be read, `GetPicture` returns `NULL`.

Discussion

The `GetPicture` function calls the Resource Manager function `GetResource` as follows:

```
GetResource('PICT', picID)
```

Special Considerations

To release the memory occupied by a picture stored in a `'PICT'` resource, use the Resource Manager function `ReleaseResource`.

The `GetPicture` function may move or purge memory.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetPixBaseAddr

Obtains a pointer to an offscreen pixel map.

```
Ptr GetPixBaseAddr (
   PixMapHandle pm
);
```

Parameters

pm

A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` (page 2642) function.

Return Value

A 32-bit pointer to the beginning of a pixel image. If the offscreen buffer has been purged, `GetPixBaseAddr` returns `NULL`.

Discussion
The `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle instead of a pointer, which is what the `baseAddr` field for an onscreen pixel map contains. You must use the `GetPixBaseAddr` function to obtain a pointer to the `PixMap` structure for an offscreen graphics world.

Your application should never directly access the `baseAddr` field of the `PixMap` structure for an offscreen graphics world; instead, always use `GetPixBaseAddr`. If your application is using 24-bit mode, use the `PixMap32Bit` (page 2749) function to determine whether a pixel map requires 32-bit addressing mode for access to its pixel image.

Special Considerations

Any QuickDraw functions that your application uses after calling `GetPixBaseAddr` may change the base address for the offscreen pixel image.

The `GetPixBaseAddr` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
ASCIIMoviePlayerSample
QTCarbonShell

Declared In
`QDOffscreen.h`

GetPixBounds

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Rect * GetPixBounds (
   PixMapHandle pixMap,
   Rect *bounds
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPixDepth

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short GetPixDepth (
   PixMapHandle pixMap
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetPixel

Determines whether the pixel associated with a point is black or white. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean GetPixel (
    short h,
    short v
);
```

Parameters

h

The horizontal coordinate of the point for the pixel to be tested.

v

The vertical coordinate of the point for the pixel to be tested.

Return Value

Returns `TRUE` if the pixel is black or `FALSE` if it is white.

Discussion

The selected pixel is immediately below and to the right of the point whose coordinates you supply in the `h` and `v` parameters, in the local coordinates of the current graphics port. There's no guarantee that the specified pixel actually belongs to the current graphics port, however it may have been drawn in a graphics port overlapping the current one. To see if the point indeed belongs to the current graphics port, you could use the `PtInRgn` function to test whether the point is in the current graphics port's visible region.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetPixelsState

Saves the current information about the memory allocated for an offscreen pixel image. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GWorldFlags GetPixelsState (
   PixMapHandle pm
);
```

Parameters

pm

A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` (page 2642) function.

Return Value

Information about the memory allocated for the base address for an offscreen pixel image. This result can be either of the constants, `pixelsPurgeable` or `pixelsLocked`. If the `pixelsPurgeable` flag is not returned, then the base address for the offscreen pixel image is unpurgeable. If the `pixelsLocked` flag is not returned, then the base address for the offscreen pixel image is unlocked.

Discussion

After using `GetPixelsState` to save this state information, use the `SetPixelsState` (page 2799) function to restore this state to the offscreen graphics world.

After using `GetPixelsState` and before using `SetPixelsState`, temporarily use the `AllowPurgePixels` (page 2573) function to make the base address for an offscreen pixel image purgeable, the `NoPurgePixels` (page 2729) function to make it unpurgeable, the `LockPixels` (page 2704) function to prevent it from being moved, and the `UnlockPixels` (page 2829) function to allow it to be moved.

Special Considerations

The `GetPixelsState` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QDOffscreen.h`

GetPixPat

Obtains a pixel pattern (`'ppat'`) resource stored in a resource file. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixPatHandle GetPixPat (
   short patID
);
```

Parameters

patID

The resource ID for a resource of type `'ppat'`.

Return Value
A handle to the pixel pattern having the resource ID you specify in the `patID` parameter. The `GetPixPat` function calls the following Resource Manager function with these parameters:

```
GetResource('ppat', patID);
```

If a `'ppat'` resource with the ID that you request does not exist, the `GetPixPat` function returns `NULL`.

Discussion
When you are finished with the pixel pattern, use the `DisposePixPat` (page 2602) function. For more information on the pixel pattern resource, see `'ppat'`.

Pixel patterns can use colors at any pixel depth and can be of any width and height that's a power of 2. To create a pixel pattern, you typically define it in a `'ppat'` resource, which you store in a resource file. To retrieve the pixel pattern stored in a `'ppat'` resource, you can use the `GetPixPat` function.

Special Considerations

The `GetPixPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPixRowBytes

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
SInt32 GetPixRowBytes (
   PixMapHandle pm
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

GetPort

Saves the current graphics port (basic or color). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetPort (
   GrafPtr *port
);
```

Parameters

port

On return, a pointer to a `GrafPort` structure for the current graphics port. If the current graphics port is a color graphics port, `GetPort` coerces its `CGrafPort` structure into a `GrafPort` structure.

Discussion

When your application runs in Color QuickDraw or uses offscreen graphics worlds, it should use the `GetGWorld` function instead of `GetPort`. The `GetGWorld` function saves the current graphics port for basic and color graphics ports as well as offscreen graphics worlds.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`QuickdrawAPI.h`

GetPortBackColor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RGBColor * GetPortBackColor (
   CGrafPtr port,
   RGBColor *backColor
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetPortBackPixPat

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixPatHandle GetPortBackPixPat (
   CGrafPtr port,
   PixPatHandle backPattern
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortBitMapForCopyBits

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
const BitMap * GetPortBitMapForCopyBits (
   CGrafPtr port
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortBounds

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Rect * GetPortBounds (
   CGrafPtr port,
   Rect *rect
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`QuickdrawAPI.h`

GetPortChExtra

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short GetPortChExtra (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortClipRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RgnHandle GetPortClipRegion (
   CGrafPtr port,
   RgnHandle clipRgn
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortCustomXFerProc

(Deprecated in Mac OS X v10.4.)

Not recommended

```
OSErr GetPortCustomXFerProc (
   CGrafPtr port,
   CustomXFerProcPtr *proc,
   UInt32 *flags,
   UInt32 *refCon
);
```

Return Value
A result code.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortFillPixPat

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixPatHandle GetPortFillPixPat (
   CGrafPtr port,
   PixPatHandle fillPattern
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortForeColor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RGBColor * GetPortForeColor (
   CGrafPtr port,
   RGBColor *foreColor
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortFracHPenLocation

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short GetPortFracHPenLocation (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortGrafProcs

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CQDProcsPtr GetPortGrafProcs (
   CGrafPtr port
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortHiliteColor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RGBColor * GetPortHiliteColor (
   CGrafPtr port,
   RGBColor *hiliteColor
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortOpColor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RGBColor * GetPortOpColor (
   CGrafPtr port,
   RGBColor *opColor
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortPenLocation

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Point * GetPortPenLocation (
   CGrafPtr port,
   Point *penLocation
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortPenMode

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
SInt32 GetPortPenMode (
   CGrafPtr port
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortPenPixPat

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixPatHandle GetPortPenPixPat (
   CGrafPtr port,
   PixPatHandle penPattern
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortPenSize

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Point * GetPortPenSize (
   CGrafPtr port,
   Point *penSize
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortPenVisibility

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short GetPortPenVisibility (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortPixMap

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixMapHandle GetPortPixMap (
   CGrafPtr port
);
```

Return Value

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortSpExtra

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Fixed GetPortSpExtra (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortTextFace

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Style GetPortTextFace (
   CGrafPtr port
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortTextFont

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short GetPortTextFont (
   CGrafPtr port
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetPortTextMode

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short GetPortTextMode (
   CGrafPtr port
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GetPortTextSize

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short GetPortTextSize (
   CGrafPtr port
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetPortVisibleRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RgnHandle GetPortVisibleRegion (
   CGrafPtr port,
   RgnHandle visRgn
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetQDGlobalsArrow

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Cursor * GetQDGlobalsArrow (
   Cursor *arrow
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetQDGlobalsBlack

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Pattern * GetQDGlobalsBlack (
   Pattern *black
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetQDGlobalsDarkGray

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Pattern * GetQDGlobalsDarkGray (
   Pattern *dkGray
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetQDGlobalsGray

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Pattern * GetQDGlobalsGray (
   Pattern *gray
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetQDGlobalsLightGray

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Pattern * GetQDGlobalsLightGray (
   Pattern *ltGray
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

GetQDGlobalsRandomSeed

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
long GetQDGlobalsRandomSeed (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

GetQDGlobalsScreenBits

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
BitMap * GetQDGlobalsScreenBits (
   BitMap *screenBits
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

GetQDGlobalsThePort

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
CGrafPtr GetQDGlobalsThePort (
   void
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetQDGlobalsWhite

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Pattern * GetQDGlobalsWhite (
   Pattern *white
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetRegionBounds

```
Rect * GetRegionBounds (
   RgnHandle region,
   Rect *bounds
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GetSubTable

Searches one color table for the best matches to colors in another color table. Your application should not need to call this function; it is used by system software only. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GetSubTable (
   CTabHandle myColors,
   short iTabRes,
   CTabHandle targetTbl
);
```

Parameters

myColors

A handle to a color table containing the colors for which you want matches.

iTabRes

The resolution of the inverse table to be used.

targetTbl

A handle to a color table whose colors are to be matched. If you supply `NULL` for `targetTbl`, then the Color Manager searches the current `GDevice` data structure's CLUT, and uses its inverse table. Otherwise a temporary inverse table is built, with a resolution of the value in the `iTabRes` parameter.

Discussion

The Color Manager uses the `Color2Index` (page 2583) function for each `RGBColor` data structure in the color table of the `myColors` parameter. It determines the best match in the target table and stores that index value in the `value` field of the color table of the `myColors` parameter.

Depending on the requested resolution, building the inverse table can require large amounts of temporary space in the application heap: twice the size of the table itself, plus a fixed overhead of 3–15 KB for each inverse table resolution.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

GlobalToLocal

Converts the coordinates of a point from global coordinates to the local coordinates of the current graphics port (basic or color). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GlobalToLocal (
   Point *pt
);
```

Parameters

pt

A pointer to a point expressed in global coordinates (where the upper-left corner of the main screen has coordinates [0,0]). On return, this point is converted to local coordinates.

Discussion
The GlobalToLocal procedure takes a point expressed in global coordinates (where the upper-left corner of the main screen has coordinates [0,0]) and converts it into the local coordinates of the current graphics port.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

GrafDevice

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void GrafDevice (
   short device
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

HandleToRgn

```
void HandleToRgn (
   Handle oldRegion,
   RgnHandle region
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

HideCursor

Hides the cursor if it is visible on the screen.

```
void HideCursor (
   void
);
```

Discussion
The `HideCursor` function removes the cursor from the screen, restores the bits under the cursor image, and decrements the cursor level (which `InitCursor` initialized to 0). You might want to use HideCursor when the user is using the keyboard to create content in one of your application's windows. Every call to `HideCursor` should be balanced by a subsequent call to the `ShowCursor` (page 2813) function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`QuickdrawAPI.h`

HidePen

Makes the graphics pen invisible, so that pen drawing doesn't show on the screen. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void HidePen (
   void
);
```

Discussion
The `HidePen` function is called by the `OpenRgn` (page 2737) , `OpenPicture`, and `OpenPoly` (page 2737) functions so that you can create regions, pictures, and polygons without drawing on the screen.

The `HidePen` function decrements the `pnVis` field of the current graphics port. The `pnVis` field is initialized to 0 by the `OpenPort` function. Whenever `pnVis` is negative, the pen does not draw on the screen. The `pnVis` field keeps track of the number of times the pen has been hidden to compensate for nested calls to the `HidePen` and `ShowPen` functions.

Every call to `HidePen` should be balanced by a subsequent call to `ShowPen` (page 2814).

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

HiliteColor

Changes the highlight color for the current color graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void HiliteColor (
   const RGBColor *color
);
```

Parameters

color

An `RGBColor` structure that defines the highlight color.

Discussion

All drawing operations that use the `hilite` transfer mode use the highlight color. When a color graphics port is created, its highlight color is initialized from the global variable `HiliteRGB`.

If the current graphics port is a basic graphics port, `HiliteColor` has no effect.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

Index2Color

Obtains the `RGBColor` data structure corresponding to an index value in the color table of the current `GDevice` data structure. Your application should not need to call this function; it is used by system software only. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void Index2Color (
   long index,
   RGBColor *aColor
);
```

Parameters

index

The index value whose color entry is sought; you should supply a long integer in which the high-order word is padded with zeros.

aColor

A pointer to the returned `RGBColor` data structure.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

InitCursor

Sets the cursor to the standard arrow and makes the cursor visible.

```
void InitCursor (
   void
);
```

Discussion
This function initializes the standard arrow cursor, sets the current cursor to the standard arrow, and makes the cursor visible. Classic Mac OS applications need to call this function when launching because the system sets the cursor to the watch cursor. Carbon applications running in Mac OS 9 or Mac OS X do not need to call this function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
HideMenuBar
ictbSample
Simple DrawSprocket

Declared In
`QuickdrawAPI.h`

InitGDevice

Initializes a `GDevice` structure. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InitGDevice (
   short qdRefNum,
   long mode,
   GDHandle gdh
);
```

Parameters

qdRefNum
Reference number of the graphics device. System software sets this number at system startup time for most graphics devices.

mode
The device configuration mode. Used by the screen driver, this value sets the pixel depth and specifies color or black and white.

gdh
The handle, returned by the `NewGDevice` function, to the `GDevice` (page 2859) structure to be initialized.

Discussion
The `InitGDevice` function sets the graphics device whose driver has the reference number specified in the `gdRefNum` parameter to the mode specified in the `mode` parameter. The `InitGDevice` function then fills out the `GDevice` structure, previously created with the `NewGDevice` function, to contain all information describing that mode.

The `mode` parameter determines the configuration of the device. Possible modes for a device are determined by interrogating the video device's ROM through Slot Manager functions. The information describing the device's mode is primarily contained in the video device's ROM. If the video device has a fixed color table, then that table is read directly from the ROM. If the video device has a variable color table, then `InitGDevice` uses the default color table defined in a `'clut'` resource, contained in the System file, that has a resource ID equal to the video device's pixel depth.

In general, your application should never need to call `InitGDevice`. All video devices are initialized at start time, and users change modes through the Monitors control panel.

If your program uses `NewGDevice` to create a graphics device without a driver, `InitGDevice` does nothing; instead, your application must initialize all fields of the `GDevice` structure. After your application initializes the color table for the `GDevice` structure, call the Color Manager function `MakeITable` to build the inverse table for the graphics device.

Special Considerations

The `InitGDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

InsetRect

Shrinks or expands a rectangle.

```
void InsetRect (
    Rect * r,
    short dh,
    short dv
);
```

Parameters

r

A pointer to the rectangle to alter.

dh

The horizontal distance to move the left and right sides in toward or outward from the center of the rectangle.

dv

The vertical distance to move the top and bottom sides in toward or outward from the center of the rectangle.

Discussion

The `InsetRect` function shrinks or expands the rectangle that you specify in the `r` parameter: the left and right sides are moved in by the amount you specify in the `dh` parameter; the top and bottom are moved toward the center by the amount you specify in the `dv` parameter. If the value you pass in `dh` or `dv` is negative,

the appropriate pair of sides is moved outward instead of inward. The effect is to alter the size by `2*dh` horizontally and `2*dv` vertically, with the rectangle remaining centered in the same place on the coordinate plane.

If the resulting width or height becomes less than 1, the rectangle is set to the empty rectangle (0,0,0,0).

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawAPI.h`

InsetRgn

Shrinks or expands a region.

```
void InsetRgn (
   RgnHandle rgn,
   short dh,
   short dv
);
```

Parameters

rgn

A handle to the region to alter.

dh

The horizontal distance to move points on the left and right boundaries in toward or outward from the center.

dv

The vertical distance to move points on the top and bottom boundaries in toward or outward from the center.

Discussion
The `InsetRgn` function moves all points on the region boundary of the region whose handle you pass in the `rgn` parameter inward by the vertical distance that you specify in the `dv` parameter and by the horizontal distance that you specify in the `dh` parameter. If you specify negative values for `dh` or `dv`, the `InsetRgn` function moves the points outward in that direction.

The `InsetRgn` function leaves the region's center at the same position, but moves the outline in (for positive values of `dh` and `dv`) or out (for negative values of `dh` and `dv`). Using `InsetRgn` on a rectangular region has the same effect as using the `InsetRect` function.

Special Considerations

The `InsetRgn` function temporarily uses heap space that's twice the size of the original region.

The `InsetRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

InvertArc

Inverts the pixels of a wedge. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvertArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

r
The rectangle that defines an oval's boundaries.

startAngle
The angle indicating the start of the arc.

arcAngle
The angle indicating the arc's extent.

Discussion
The `InvertArc` function inverts the pixels enclosed by a wedge of the oval bounded by the rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. As in `FrameArc` (page 2629) , use the `startAngle` and `arcAngle` parameters to define the arc of the wedge.

This function leaves the location of the graphics pen unchanged.

Special Considerations

The `InvertArc` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct devices or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

InvertColor

Finds the complement of an `RGBColor` data structure. This function is used only by system software. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvertColor (
   RGBColor *myColor
);
```

Parameters

myColor

A pointer to the `RGBColor` data structure for which the complement is to be found. The `InvertColor` function returns the complement of an absolute color, using the list of complement functions in the current device data structure. The default complement function uses the one's complement of each component of the given color.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

InvertOval

Inverts the pixels enclosed by an oval. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvertOval (
   const Rect *r
);
```

Parameters

r

The rectangle that defines the oval's boundary.

Discussion
The `InvertOval` function inverts the pixels enclosed by an oval just inside the bounding rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. The pen location does not change.

Special Considerations

The `InvertOval` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct devices or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

InvertPoly

Inverts the pixels enclosed by a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvertPoly (
   PolyHandle poly
);
```

Parameters

poly

A handle to a polygon, the pixels of which you want to invert. The `OpenPoly` (page 2737) function returns this handle when you first create the polygon.

Discussion

The `InvertPoly` function inverts the pixels enclosed by the polygon whose handle you pass in the `poly` parameter. Every white pixel becomes black and every black pixel becomes white.

This function leaves the location of the graphics pen unchanged.

`InvertPoly` temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 2707) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Special Considerations

The `InvertPoly` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with 1-bit or direct pixels. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

InvertRect

Inverts the pixels enclosed by a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvertRect (
    const Rect * r
);
```

Parameters

r

The rectangle whose enclosed pixels are to be inverted.

Discussion
The `InvertRect` function inverts the pixels enclosed by the rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. The pen location does not change.

Special Considerations

The `InvertRect` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct pixels or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

InvertRgn

Inverts the pixels enclosed by a region. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvertRgn (
    RgnHandle rgn
);
```

Parameters

rgn
A handle to the region whose pixels are to invert.

Discussion
The `InvertRgn` function inverts the pixels enclosed by the region whose handle you pass in the `rgn` parameter. Every white pixel becomes black and every black pixel becomes white.

This function leaves the location of the graphics pen unchanged.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Special Considerations

The `InvertRgn` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with 1-bit or direct pixels. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

InvertRoundRect

Inverts the pixels enclosed by a rounded rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvertRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

r
The rectangle that defines the rounded rectangle's boundaries.

ovalWidth
The width of the oval defining the rounded corner.

ovalHeight
The height of the oval defining the rounded corner.

Discussion
The `InvertRoundRect` function inverts the pixels enclosed by the rounded rectangle bounded by the rectangle that you specify in the `r` parameter. Every white pixel becomes black and every black pixel becomes white. The `ovalWidth` and `ovalHeight` parameters specify the diameters of curvature for the corners. The pen location does not change.

Special Considerations

The `InvertRoundRect` function was designed for 1-bit images in basic graphics ports. This function operates on color pixels in color graphics ports, but the results are predictable only with direct devices or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the CLUT. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Inversion works better for direct pixels. Inverting a pure green, for example, that has red, green, and blue component values of $0000, $FFFF, and $0000 results in magenta, which has component values of $FFFF, $0000, and $FFFF.

The `InvertRoundRect` function may move or purge memory blocks in the application; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

InvokeColorComplementUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean InvokeColorComplementUPP (
   RGBColor *rgb,
   ColorComplementUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeColorSearchUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean InvokeColorSearchUPP (
   RGBColor *rgb,
   long *position,
   ColorSearchUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeDeviceLoopDrawingUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeDeviceLoopDrawingUPP (
   short depth,
   short deviceFlags,
   GDHandle targetDevice,
   SRefCon userData,
   DeviceLoopDrawingUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeDragGrayRgnUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeDragGrayRgnUPP (
   DragGrayRgnUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDArcUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDArcUPP (
   GrafVerb verb,
   const Rect *r,
   short startAngle,
   short arcAngle,
   QDArcUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDBitsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDBitsUPP (
   const BitMap *srcBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short mode,
   RgnHandle maskRgn,
   QDBitsUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDCommentUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDCommentUPP (
   short kind,
   short dataSize,
   Handle dataHandle,
   QDCommentUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDGetPicUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDGetPicUPP (
   void *dataPtr,
   short byteCount,
   QDGetPicUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeQDJShieldCursorUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDJShieldCursorUPP (
   short left,
   short top,
   short right,
   short bottom,
   QDJShieldCursorUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeQDLineUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDLineUPP (
   Point newPt,
   QDLineUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeQDOpcodeUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDOpcodeUPP (
   const Rect *fromRect,
   const Rect *toRect,
   UInt16 opcode,
   SInt16 version,
   QDOpcodeUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDOvalUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDOvalUPP (
   GrafVerb verb,
   const Rect *r,
   QDOvalUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDPolyUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDPolyUPP (
   GrafVerb verb,
   PolyHandle poly,
   QDPolyUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDPutPicUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDPutPicUPP (
   const void *dataPtr,
   short byteCount,
   QDPutPicUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeQDRectUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDRectUPP (
   GrafVerb verb,
   const Rect *r,
   QDRectUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeQDRgnUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDRgnUPP (
   GrafVerb verb,
   RgnHandle rgn,
   QDRgnUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeQDRRectUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDRRectUPP (
   GrafVerb verb,
   const Rect *r,
   short ovalWidth,
   short ovalHeight,
   QDRRectUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDStdGlyphsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus InvokeQDStdGlyphsUPP (
   void *dataStream,
   ByteCount size,
   QDStdGlyphsUPP userUPP
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

InvokeQDTextUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void InvokeQDTextUPP (
   short byteCount,
   const void *textBuf,
   Point numer,
   Point denom,
   QDTextUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeQDTxMeasUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short InvokeQDTxMeasUPP (
   short byteCount,
   const void *textAddr,
   Point *numer,
   Point *denom,
   FontInfo *info,
   QDTxMeasUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

InvokeRegionToRectsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus InvokeRegionToRectsUPP (
   UInt16 message,
   RgnHandle rgn,
   const Rect *rect,
   void *refCon,
   RegionToRectsUPP userUPP
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawAPI.h

IsPortClipRegionEmpty

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsPortClipRegionEmpty (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsPortColor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsPortColor (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsPortOffscreen

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsPortOffscreen (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsPortPictureBeingDefined

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsPortPictureBeingDefined (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsPortPolyBeingDefined

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsPortPolyBeingDefined (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsPortRegionBeingDefined

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsPortRegionBeingDefined (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsPortVisibleRegionEmpty

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsPortVisibleRegionEmpty (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsRegionRectangular

```
Boolean IsRegionRectangular (
   RgnHandle region
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsValidPort

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean IsValidPort (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

IsValidRgnHandle

```
Boolean IsValidRgnHandle (
   RgnHandle rgn
);
```

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

KillPicture

Releases the memory occupied by a picture not stored in a `'PICT'` resource. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void KillPicture (
   PicHandle myPicture
);
```

Parameters

myPicture
A handle to the picture whose memory can be released.

Discussion
Use this function only when you are completely finished with a picture.

Special Considerations

If you use the Window Manager function `SetWindowPic` to store a picture handle in the window structure, use the Window Manager function `DisposeWindow` or `CloseWindow` to release the memory allocated to the picture. These functions automatically call `KillPicture` for the picture.

If the picture is stored in a `'PICT'` resource, use the Resource Manager function `ReleaseResource` instead of `KillPicture`. The Window Manager functions `DisposeWindow` and `CloseWindow` will not delete it. Instead, call `ReleaseResource` before calling `DisposeWindow` or `CloseWindow`.

The `KillPicture` function may move or purge memory.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

KillPoly

Releases the memory occupied by a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void KillPoly (
   PolyHandle poly
);
```

Parameters

poly
A handle to the polygon to dispose of.

Discussion
Use `KillPoly` only when you are completely through with a polygon.

Special Considerations

The `KillPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

Line

Draws a line a specified distance from the graphics pen's current location in the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void Line (
   short dh,
   short dv
);
```

Parameters

dh

The horizontal distance of the graphics pen's movement.

dv

The vertical distance of the graphics pen's movement.

Discussion

Starting at the current location of the graphics pen, the `Line` function draws a line the horizontal distance that you specify in the `dh` parameter and the vertical distance that you specify in the `dv` parameter. The `Line` function calls

LineTo(h+dh,v+dv)

where `(h,v)` is the current location in local coordinates. The pen location becomes the coordinates of the end of the line after the line is drawn. If you are using `Line` to draw a region or polygon, its outline is infinitely thin and is not affected by the values of the `pnSize`, `pnMode`, and `pnPat` fields of the graphics port.

Special Considerations

The `Line` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

LineTo

Draws a line from the graphics pen's current location to a new location. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LineTo (
    short h,
    short v
);
```

Parameters

h

The horizontal coordinate of the graphics pen's new location.

v

The vertical coordinate of the graphics pen's new location.

Discussion

The `LineTo` function draws a line from the graphics pen's current location in the current graphics port to the new location `(h,v)`, which you specify in the local coordinates of the current graphics port. If you are using `LineTo` to draw a region or polygon, its outline is infinitely thin and is not affected by the values of the `pnSize`, `pnMode`, or `pnPat` field of the graphics port.

Special Considerations

The `LineTo` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

LMGetCursorNew

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean LMGetCursorNew (
   void
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

LMGetDeviceList

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle LMGetDeviceList (
   void
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

LMGetFractEnable

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
UInt8 LMGetFractEnable (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

LMGetHiliteMode

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
UInt8 LMGetHiliteMode (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

LMGetHiliteRGB

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMGetHiliteRGB (
   RGBColor *hiliteRGBValue
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMGetLastFOND

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Handle LMGetLastFOND (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMGetLastSPExtra

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
SInt32 LMGetLastSPExtra (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMGetMainDevice

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle LMGetMainDevice (
   void
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMGetQDColors

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Handle LMGetQDColors (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMGetScrHRes

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
SInt16 LMGetScrHRes (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMGetScrVRes

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
SInt16 LMGetScrVRes (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

LMGetTheGDevice

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle LMGetTheGDevice (
   void
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

LMGetWidthListHand

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Handle LMGetWidthListHand (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

LMGetWidthPtr

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Ptr LMGetWidthPtr (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMGetWidthTabHandle

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Handle LMGetWidthTabHandle (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetCursorNew

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetCursorNew (
   Boolean value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetDeviceList

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetDeviceList (
   GDHandle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetFractEnable

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetFractEnable (
   UInt8 value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetHiliteMode

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetHiliteMode (
   UInt8 value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetHiliteRGB

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetHiliteRGB (
   const RGBColor *hiliteRGBValue
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetLastFOND

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetLastFOND (
   Handle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetLastSPExtra

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetLastSPExtra (
   SInt32 value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetMainDevice

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetMainDevice (
   GDHandle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetQDColors

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetQDColors (
   Handle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetScrHRes

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetScrHRes (
   SInt16 value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetScrVRes

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetScrVRes (
   SInt16 value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetTheGDevice

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetTheGDevice (
   GDHandle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetWidthListHand

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetWidthListHand (
   Handle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetWidthPtr

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetWidthPtr (
   Ptr value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LMSetWidthTabHandle

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LMSetWidthTabHandle (
   Handle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LocalToGlobal

Converts a point's coordinates from the local coordinates of the current graphics port (basic or color) to global coordinates. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void LocalToGlobal (
   Point *pt
);
```

Parameters

pt
A pointer to a point in local coordinates. On return, this point is converted to global coordinates.

Discussion
The `LocalToGlobal` function converts the given point from the current graphics port's local coordinate system into the global coordinate system (where the upper-left corner of the main screen has coordinates [0,0]). This global point can then be compared to other global points, or it can be changed into the local coordinates of another graphics port.

Because a rectangle is defined by two points, you can convert a rectangle into global coordinates with two calls to `LocalToGlobal`. In conjunction with `LocalToGlobal`, you can use the `OffsetRect`, `OffsetRgn`, or `OffsetPoly` functions to convert a rectangle, region, or polygon into global coordinates.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

LockPixels

Prevents the base address for an offscreen pixel image from being moved while you draw into or copy from its pixel map. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean LockPixels (
   PixMapHandle pm
);
```

Parameters

pm

A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` (page 2642) function .

Return Value
If the base address for an offscreen pixel image hasn't been purged by the Memory Manager or is not purgeable, `LockPixels` returns `TRUE` as its function result, and your application can draw into or copy from the offscreen pixel map. However, if the base address for an offscreen pixel image has been purged, `LockPixels` returns `FALSE` to indicate that you can perform no drawing to or copying from the pixel map. At that point, your application should either call the `UpdateGWorld` (page 2831) function to reallocate the offscreen pixel image and then reconstruct it, or draw directly in a window instead of preparing the image in an offscreen graphics world.

Discussion
You must call `LockPixels` before drawing to or copying from an offscreen graphics world.

The `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle instead of a pointer (which is what the `baseAddr` field for an onscreen pixel map contains). The `LockPixels` function dereferences the `PixMap` handle into a pointer. When you use the `UnlockPixels` function the handle is recovered.

As soon as you are finished drawing into and copying from the offscreen pixel image, call the `UnlockPixels` (page 2829) function.

Special Considerations

The `LockPixels` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
ASCIIMoviePlayerSample
QTCarbonShell
WhackedTV

Declared In
QDOffscreen.h

LockPortBits

Acquires an exclusive lock on the back buffer for a Carbon window. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr LockPortBits (
   GrafPtr port
);
```

Parameters

port
A window port.

Return Value
A result code. If `noErr`, the window's back buffer is locked and available for direct access.

Discussion
In Mac OS X, a Carbon window's port bits are in a back buffer shared by the application and the Quartz compositor (sometimes called the window server). When an application needs to update this buffer, the Quartz compositor must be locked out temporarily. You can use this function together with `UnlockPortBits` (page 2829) to acquire and release an exclusive lock.

If you're using QuickDraw or Quartz 2D to draw in a window, you do not need to call this function—buffer locks are handled for you automatically. If you're writing code that reads or modifies the port bits directly, you should bracket your code with calls to this function and `UnlockPortBits` (page 2829).

Nested calls to this function for the same port are permitted. For a given port, if you call `LockPortBits` *n* times, the lock is actually released after the *n*th balancing call to `UnlockPortBits` (page 2829).

You should not call any QuickTime functions while holding the lock. To avoid degrading the user experience, you should release the lock as quickly as possible.

In Mac OS 9, this function does nothing and returns `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
QuickdrawAPI.h

MakeITable

Generates an inverse table for a color table. Your application should not need to call this function; it is used by system software only. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void MakeITable (
   CTabHandle cTabH,
   ITabHandle iTabH,
   short res
);
```

Parameters

cTabH

The color table for which an inverse table is to be generated. Passing `NULL` substitutes an appropriate handle from the current `GDevice` data structure.

iTabH

The generated inverse table. Passing `NULL` substitutes an appropriate handle from the current `GDevice` data structure.

res

The resolution needed for the inverse table. Passing 0 substitutes the current `GDevice` data structure's inverse table resolution.

Discussion

The `MakeITable` function generates an inverse table based on the current contents of the color table pointed to by the `cTabH` parameter, with a resolution specified by the value in the `res` parameter. Reserved color table pixel values are not included in the resulting color table. `MakeITable` tests its input parameters and returns an error in `QDError` if the resolution is less than three or greater than five.

This function allows maximum precision in mapping colors, even if colors in the color table differ by less than the resolution of the inverse table. Five-bit inverse tables are not needed when drawing in normal Color QuickDraw modes. However, Color QuickDraw transfer modes such as add, subtract, and blend may require a 5-bit inverse table for best results with certain color tables. The `'mitq'` resource governs how much memory is allocated for temporary internal structures; this resource type is for internal use only.

Depending on the requested resolution, building the inverse table can require large amounts of temporary space in the application heap: twice the size of the table itself, plus a fixed overhead of 3–15 KB for each inverse table resolution.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

MakeRGBPat

Creates the appearance of otherwise unavailable colors on indexed devices. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void MakeRGBPat (
   PixPatHandle pp,
   const RGBColor *myColor
);
```

Parameters

pp

On return, a handle to the generated pixel pattern.

myColor

An `RGBColor` structure that defines the color you want to approximate.

Discussion

The `MakeRGBPat` function generates a `PixPat` (page 2871) structure that approximates the color you specify in the `myColor` parameter. For example, if your application draws to an indexed device that supports 4 bits per pixel, you only have 16 colors available if you simply set the foreground color and draw. If you use `MakeRGBPat` to create a pattern, and then draw using that pattern, you effectively get 125 different colors. If the graphics device has 8 bits per pixel, you effectively get 2197 colors. (More color are theoretically possible; this implementation opted for a fast pattern selection rather than the best possible pattern selection.)

For a pixel pattern, the (** `patMap).bounds` field of the `PixPat` structure always contains the values (0,0,8,8), and the (** `patMap).rowbytes` field equals 2.

Because patterns produced with `MakeRGBPat` aren't usually solid—they provide a selection of colors by alternating between colors, with up to four colors in a pattern— lines that are only one pixel wide may not look good.

When `MakeRGBPat` creates a `ColorTable` structure, it fills in only the `rgb` fields of its `ColorSpec` structures; the `value` fields are computed at the time the drawing actually takes place, using the current pixel depth for the system.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

MapPoly

Maps and scales a polygon within one rectangle to another rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void MapPoly (
   PolyHandle poly,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

poly

A handle to a polygon. Upon input, this is the polygon to map. Upon completion, this polygon is the one mapped to a new location.

srcRect
The rectangle containing the polygon.

dstRect
The rectangle in which the new region will be mapped.

Discussion
The `MapPoly` function takes a polygon within one rectangle and maps and scales it to another rectangle. In the `poly` parameter, you specify a handle to a polygon that lies within the rectangle that you specify in the `srcRect` parameter. By calling the `MapPt` function to map all the points that define the polygon specified in the `poly` parameter, `MapPoly` maps and scales it to the rectangle that you specify in the `dstRect` parameter. The `MapPoly` function returns the result in the polygon whose handle you initially passed in the `poly` parameter.

Similar to the `MapRgn` function described in the previous section, the `MapPoly` function is useful for determining whether a polygon operation will exceed available memory.

Special Considerations

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

MapPt

Maps a point in one rectangle to an equivalent position in another rectangle.

```
void MapPt (
   Point *pt,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

pt
Upon input, a pointer to the point in the source rectangle to map; upon completion, a pointer to its mapped position in the destination rectangle.

srcRect
The source rectangle containing the original point.

dstRect
The destination rectangle in which the point will be mapped.

Discussion
The `MapPt` function maps a point in one rectangle to an equivalent position in another rectangle.

In the `pt` parameter, you specify a point that lies within the rectangle that you specify in the `srcRect` parameter. The `MapPt` function maps this point to a similarly located point within the rectangle that you specify in the `dstRect` parameter—that is, to where it would fall if it were part of a drawing being expanded or shrunk to fit the destination rectangle. The `MapPt` function returns the location of the mapped point in the `pt` parameter. For example, a corner point of the source rectangle would be mapped to the corresponding corner point of the destination rectangle in `dstRect`, and the center of the source rectangle would be mapped to the center of destination rectangle.

The source and destination rectangles may overlap, and the point you specify need not actually lie within the source rectangle.

If you are going to draw inside the destination rectangle, you'll probably also want to scale the graphics pen size accordingly with `ScalePt` (page 2782).

Special Considerations

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawAPI.h`

MapRect

Maps and scales a rectangle within one rectangle to another rectangle.

```
void MapRect (
   Rect *r,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

r
Upon input, a pointer to the rectangle to map; upon completion, the mapped rectangle.

srcRect
The rectangle containing the rectangle to map.

dstRect
The rectangle in which the new rectangle will be mapped.

Discussion
The `MapRect` function takes a rectangle within one rectangle and maps and scales it to another rectangle. In the `r` parameter, you specify a rectangle that lies within the rectangle that you specify in the `srcRect` parameter. By calling the `MapPt` function to map the upper-left and lower-right corners of the rectangle in the `r` parameter, `MapRect` maps and scales it to the rectangle that you specify in the `dstRect` parameter. The `MapRect` function returns the newly mapped rectangle in the `r` parameter.

Special Considerations

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawAPI.h`

MapRgn

Maps and scales a region within one rectangle to another rectangle.

```
void MapRgn (
   RgnHandle rgn,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

rgn

A handle to a region. Upon input, this is the region to map. Upon completion, this region is the one mapped to a new location.

srcRect

The rectangle containing the region to map.

dstRect

The rectangle in which the new region will be mapped.

Discussion

The `MapRgn` function takes a region within one rectangle and maps and scales it to another rectangle. In the `rgn` parameter, you specify a handle to a region that lies within the rectangle that you specify in the `srcRect` parameter. By calling the `MapPt` function to map all the points of the region in the `rgn` parameter, `MapRgn` maps and scales it to the rectangle that you specify in the `dstRect` parameter. The `MapRgn` function returns the result in the region whose handle you initially passed in the `rgn` parameter.

The `MapRgn` function is useful for determining whether a region operation will exceed available memory. By mapping a large region into a smaller one and performing the operation (without actually drawing), you can estimate how much memory will be required by the anticipated operation.

Special Considerations

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

The `MapRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

Move

Moves the graphics pen a particular distance. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void Move (
   short dh,
   short dv
);
```

Parameters

dh

The horizontal distance of the graphics pen's movement.

dv

The vertical distance of the graphics pen's movement.

Discussion
The `Move` function moves the graphics pen from its current location in the current graphics port a horizontal distance that you specify in the `dh` parameter and a vertical distance that you specify in the `dv` parameter. The `Move` function calls

MoveTo(h+dh,v+dv)

where `(h,v)` is the graphics pen's current location in local coordinates. The `Move` function performs no drawing.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

MovePortTo

Changes the position of the port rectangle of the current graphics port (basic or color). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void MovePortTo (
   short leftGlobal,
   short topGlobal
);
```

Parameters

leftGlobal

The horizontal distance to move the port rectangle.

topGlobal

The vertical distance to move the port rectangle.

Discussion

The `MovePortTo` function is normally called only by the Window Manager. The `MovePortTo` function changes the position of the current graphics port's port rectangle: the `leftGlobal` and `topGlobal` parameters set the distance between the upper-left corner of the boundary rectangle and the upper-left corner of the new port rectangle.

This does not affect the screen; it merely changes the location at which subsequent drawing inside the graphics port appears. Like the `PortSize` function, `MovePortTo` doesn't change the clipping or visible region, nor does it affect the local coordinate system of the graphics port.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

MoveTo

Moves the graphics pen to a particular location in the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void MoveTo (
   short h,
   short v
);
```

Parameters

h

The horizontal coordinate of the graphics pen's new position.

v

The vertical coordinate of the graphics pen's new position.

Discussion

The `MoveTo` function changes the graphics pen's current location to the new horizontal coordinate you specify in the `h` parameter and the new vertical coordinate you specify in the `v` parameter. Specify the new location in the local coordinates of the current graphics port. The `MoveTo` function performs no drawing.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
HideMenuBar
Simple DrawSprocket

Declared In
`QuickdrawAPI.h`

NewColorComplementUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
ColorComplementUPP NewColorComplementUPP (
   ColorComplementProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

NewColorSearchUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
ColorSearchUPP NewColorSearchUPP (
   ColorSearchProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

NewDeviceLoopDrawingUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
DeviceLoopDrawingUPP NewDeviceLoopDrawingUPP (
   DeviceLoopDrawingProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

NewDragGrayRgnUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
DragGrayRgnUPP NewDragGrayRgnUPP (
   DragGrayRgnProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

NewGDevice

Creates a new `GDevice` structure. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GDHandle NewGDevice (
   short refNum,
   long mode
);
```

Parameters

refNum

Reference number of the graphics device for which you are creating a `GDevice` structure. For most video devices, this information is set at system startup.

mode

The device configuration mode. Used by the screen driver, this value sets the pixel depth and specifies color or black and white.

Return Value
A handle to the new `GDevice` (page 2859) structure. If the request is unsuccessful, `NewGDevice` returns `NULL`.

Discussion
Generally, you do not need to use `NewGDevice`, because Color QuickDraw uses this function to create `GDevice` structures for your application automatically. When the system starts up, it allocates and initializes one handle to a `GDevice` structure for each video device it finds. When you use the `NewGWorld` function, QuickDraw automatically creates a `GDevice` structure for the new offscreen graphics world.

For the graphics device whose driver is specified in the `refNum` parameter and whose mode is specified in the `mode` parameter, the `NewGDevice` function allocates a new `GDevice` structure and all of its handles, and then calls the `InitGDevice` function to initialize the structure.

`NewGDevice` allocates the new `GDevice` structure and all of its handles in the system heap, and the `NewGDevice` function sets all attributes in the `gdFlags` field of the `GDevice` structure to `FALSE`. If your application creates a `GDevice` structure, use the `SetDeviceAttribute` (page 2793) function to change the flag bits in the `gdFlags` field of the `GDevice` structure to `TRUE`. Your application should never directly change the `gdFlags` field of the `GDevice` structure. Instead, use only the `SetDeviceAttribute` function.

If your application creates a `GDevice` structure without a driver, set the `mode` parameter to –1. In this case, `InitGDevice` cannot initialize the `GDevice` structure, so your application must perform all initialization of the structure. A `GDevice` structure's default mode is defined as 128. This is assumed to be a black-and-white mode. If you specify a value other than 128 in the `mode` parameter, the structure's `gdDevType` bit in the `gdFlags` field of the `GDevice` structure is set to `TRUE` to indicate that the graphics device is capable of displaying color.

The `NewGDevice` function does not automatically insert the `GDevice` structure into the device list. In general, your application should not create `GDevice` structures, and if it ever does, it should never add them to the device list.

If your program uses `NewGDevice` to create a graphics device without a driver, `InitGDevice` does nothing; instead, your application must initialize all fields of the `GDevice` structure. After your application initializes the color table for the `GDevice` structure, call the Color Manager function `MakeITable` to build the inverse table for the graphics device.

Special Considerations

The `NewGDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NewGWorld

Creates an offscreen graphics world. (**Deprecated.** Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDErr NewGWorld (
   GWorldPtr *offscreenGWorld,
   short PixelDepth,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

offscreenGWorld

On return, a pointer to the offscreen graphics world created by this function. You use this pointer when referring to this new offscreen world in other QuickDraw functions.

PixelDepth

The pixel depth of the offscreen world; possible depths are 1, 2, 4, 8, 16, and 32 bits per pixel. The default parameter (0) uses the pixel depth of the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you specify 0 in this parameter, `NewGWorld` also uses the `GDevice` structure from this device instead of creating a new `GDevice` structure for the offscreen world. If you use `NewGWorld` on a computer that supports only basic QuickDraw, you may specify only 0 or 1 in this parameter.

boundsRect

The boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if `NewGWorld` creates one. If you specify 0 in the `pixelDepth` parameter, `NewGWorld` interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. `NewGWorld` then uses the pixel depth, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure. If you pass `NULL` in this parameter, `NewGWorld` uses the default color table for the pixel depth that you specify in the `pixelDepth` parameter. If you set the `pixelDepth` parameter to 0, `NewGWorld` ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use `NewGWorld` on a computer that supports only basic QuickDraw, you may specify only `NULL` in this parameter.

aGDevice

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case `NewGWorld` attaches this `GDevice` structure to the new offscreen graphics world. If you set the `pixelDepth` parameter to 0, or if you do not set the `noNewDevice` flag, `NewGWorld` ignores the `aGDevice` parameter, so set it to `NULL`. If you set the `pixelDepth` parameter to 0, `NewGWorld` uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NULL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

Options available to your application. You can set a combination of the flags `pixPurge`, `noNewDevice`, `useTempMem`, and `keepLocal`. If you don't wish to use any of these flags, specify 0 in this parameter to accept the default behavior for `NewGWorld`. The default behavior creates an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, it uses an existing `GDevice` structure (if you pass 0 in the `depth` parameter) or creates a new `GDevice` structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. See "Graphics World Flags" (page 2891) for a description of the values you can use here.

Return Value
A result code.

Discussion
Typically, you pass 0 in the `pixelDepth` parameter, a window's port rectangle in the `boundsRect` parameter, `NULL` in the `cTable` and `aGDevice` parameters, and in the `flags` parameter a 0. This provides your application with the default behavior of `NewGWorld`, and it supports computers running basic QuickDraw. This also allows QuickDraw to optimize the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions when your application copies the image in an offscreen graphics world into an onscreen graphics port.

The `NewGWorld` function allocates memory for an offscreen graphics port and its pixel map. On computers that support only basic QuickDraw, `NewGWorld` creates a 1-bit pixel map that your application can manipulate using other relevant functions described in this chapter. Your application can copy this 1-bit pixel map into basic graphics ports.

Unless you specify 0 in the `pixelDepth` parameter–or pass the `noNewDevice` flag in the `flags` parameter and supply a `GDevice` structure in the `aGDevice` parameter– `NewGWorld` also allocates a new offscreen `GDevice` structure.

When creating an image, use the `NewGWorld` function to create an offscreen graphics world that is optimized for an image's characteristics—for example, its best pixel depth. After creating the image, use the `CopyBits`, `CopyMask`, or `CopyDeepMask` function to copy that image to an onscreen graphics port. Color QuickDraw automatically renders the image at the best available pixel depth for the screen. Creating an image in an offscreen graphics port and then copying it to the screen in this way prevents the visual choppiness that would otherwise occur if your application were to build a complex image directly onscreen.

The `NewGWorld` function initializes the offscreen graphics port by calling the `OpenCPort` function. The `NewGWorld` function sets the offscreen graphics port's visible region to a rectangular region coincident with its boundary rectangle. The `NewGWorld` function generates an inverse table with the Color Manager function `MakeITable`, unless one of the `GDevice` structures for the screens has the same color table as the `GDevice` structure for the offscreen world, in which case `NewGWorld` uses the inverse table from that `GDevice` structure.

The address of the offscreen pixel image is not directly accessible from the `PixMap` structure for the offscreen graphics world. However, you can use the `GetPixBaseAddr` (page 2648) function to get a pointer to the beginning of the offscreen pixel image.

For purposes of estimating memory use, you can compute the size of the offscreen pixel image by using this formula:

rowBytes * (boundsRect.bottom – boundsRect.top)

In the `flags` parameter, you can specify several options. If you don't want to use any of these options, pass 0 in the `flags` parameter:

- If you specify the `pixPurge` flag, `NewGWorld` stores the offscreen pixel image in a purgeable block of memory. In this case, before drawing to or from the offscreen pixel image, your application should call the `LockPixels` (page 2704) function and ensure that it returns `TRUE`. If `LockPixels` returns `FALSE`, the memory for the pixel image has been purged, and your application should either call `UpdateGWorld` (page 2831) to reallocate it and then reconstruct the pixel image, or draw directly in a window instead of preparing the image in an offscreen graphics world. Never draw to or copy from an offscreen pixel image that has been purged without reallocating its memory and then reconstructing it.
- If you specify the `noNewDevice` flag, `NewGWorld` does not create a new offscreen `GDevice` structure. Instead, it uses the `GDevice` structure that you specify in the `aGDevice` parameter–and its associated pixel depth and color table–to create the offscreen graphics world. (If you set the `pixelDepth` parameter to 0, `NewGWorld` uses the `GDevice` structure for the screen with the greatest pixel depth among all the screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter–even if you specify the `noNewDevice` flag.) The `NewGWorld` function keeps a reference to the `GDevice` structure for the offscreen graphics world, and the `SetGWorld` (page 2796) function uses that structure to set the current graphics device.
- If you set the `useTempMem` flag, `NewGWorld` creates the base address for an offscreen pixel image in temporary memory. You generally would not use this flag, because you should use temporary memory only for fleeting purposes and only with the `AllowPurgePixels` (page 2573) function.
- If you specify the `keepLocal` flag, your offscreen pixel image is kept in Macintosh main memory and is not cached to a graphics accelerator card. use this flag carefully, as it negates the advantages provided by any graphics acceleration card that might be present.

If your application needs to change the pixel depth, boundary rectangle, or color table for an offscreen graphics world, use the `UpdateGWorld` (page 2831) function.

Special Considerations

If you supply a handle to a `ColorTable` structure in the `cTable` parameter, `NewGWorld` makes a copy of the structure and stores its handle in the offscreen `PixMap` structure. It is your application's responsibility to make sure that the `ColorTable` structure you specify in the `cTable` parameter is valid for the offscreen graphics port's pixel depth.

If when using `NewGWorld` you specify a pixel depth, color table, or `GDevice` structure that differs from those used by the window into which you copy your offscreen image, the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions require extra time to complete.

To use a custom color table in an offscreen graphics world, you need to create the associated offscreen `GDevice` structure, because Color QuickDraw needs its inverse table.

The `NewGWorld` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.k.h`

NewGWorldFromPtr

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDErr NewGWorldFromPtr (
   GWorldPtr *offscreenGWorld,
   UInt32 PixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags,
   Ptr newBuffer,
   SInt32 rowBytes
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

NewPixMap

Creates a new, initialized `PixMap` structure. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixMapHandle NewPixMap (
   void
);
```

Return Value
A handle to the new `PixMap` structure.

Discussion
All fields of the `PixMap` structure are copied from the current device's `PixMap` structure except the color table. In System 7, the `hRes` and `vRes` fields are set to 72 dpi, no matter what values the current device's `PixMap` structure contains. A handle to the color table is allocated but not initialized.

Typically, you do not need to call this function because `PixMap` structures are created for you when you create a window using the Window Manager functions `NewCWindow` and `GetNewCWindow` and when you create an offscreen graphics world with the `NewGWorld` function.

If your application creates a pixel map, your application must initialize the `PixMap` structure's color table to describe the pixels. Use the `GetCTable` (page 2636) function to read such a table from a resource file. Use the `DisposeCTable` (page 2599) function to dispose of the `PixMap` structure's color table and replace it with the one returned by `GetCTable`.

Special Considerations

The `NewPixMap` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
LiveVideoMixer2

Declared In
`QuickdrawAPI.h`

NewPixPat

Creates a new pixel pattern. Generally, however, your application should create a pixel pattern in a `'ppat'` resource. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PixPatHandle NewPixPat (
   void
);
```

Return Value
A handle to the new `PixPat` (page 2871) structure created by the `NewPixPat` function.

Discussion
This function calls the `NewPixMap` (page 2719) function to allocate the pattern's `PixMap` structure and initializes it to the same settings as the pixel map of the current `GDevice` structure. `NewPixPat` also sets the `pat1Data` field of the new `PixPat` structure to a 50 percent gray pattern. `NewPixPat` allocates new handles for the `PixPat` structure's data, expanded data, expanded map, and color table but does not initialize them instead, your application must initialize them.

Set the `rowBytes`, `bounds`, and `pixelSize` fields of the pattern's `PixMap` structure to the dimensions of the desired pattern. The `rowBytes` value should be equal to

```
(width of bounds) x pixelSize/8
```

The `rowBytes` value need not be even. The width and height of the bounds must be a power of 2. Each scan line of the pattern must be at least 1 byte in length—that is, ([width of bounds] x `pixelSize`) must be at least 8.

Your application can explicitly specify the color corresponding to each pixel value with a color table. The color table for the pattern must be placed in the `pmTable` field in the pattern's `PixMap` structure.

Including the `PixPat` structure itself, `NewPixPat` allocates a total of five handles. The sizes of the handles to the `PixPat` and `PixMap` structures are the sizes of their respective data structures. The other three handles are initially small in size. Once the pattern is drawn, the size of the expanded data is proportional to the size of the pattern data, but adjusted to the depth of the screen. The color table size is the size of the structure plus 8 bytes times the number of colors in the table.

When you are finished using the pixel pattern, use the `DisposePixPat` (page 2602) function to make the memory used by the pixel pattern available again.

Special Considerations

The `NewPixPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

NewQDArcUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDArcUPP NewQDArcUPP (
   QDArcProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

NewQDBitsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDBitsUPP NewQDBitsUPP (
   QDBitsProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawTypes.h`

NewQDCommentUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDCommentUPP NewQDCommentUPP (
   QDCommentProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDGetPicUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDGetPicUPP NewQDGetPicUPP (
   QDGetPicProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDJShieldCursorUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDJShieldCursorUPP NewQDJShieldCursorUPP (
   QDJShieldCursorProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDLineUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDLineUPP NewQDLineUPP (
   QDLineProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDOpcodeUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDOpcodeUPP NewQDOpcodeUPP (
   QDOpcodeProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDOvalUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDOvalUPP NewQDOvalUPP (
   QDOvalProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDPolyUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDPolyUPP NewQDPolyUPP (
   QDPolyProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDPutPicUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDPutPicUPP NewQDPutPicUPP (
   QDPutPicProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDRectUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDRectUPP NewQDRectUPP (
   QDRectProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDRgnUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDRgnUPP NewQDRgnUPP (
   QDRgnProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDRRectUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDRRectUPP NewQDRRectUPP (
   QDRRectProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDStdGlyphsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDStdGlyphsUPP NewQDStdGlyphsUPP (
   QDStdGlyphsProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDTextUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDTextUPP NewQDTextUPP (
   QDTextProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewQDTxMeasUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDTxMeasUPP NewQDTxMeasUPP (
   QDTxMeasProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawTypes.h

NewRegionToRectsUPP

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RegionToRectsUPP NewRegionToRectsUPP (
   RegionToRectsProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
QuickdrawAPI.h

NewRgn

Begins creating a new region.

```
RgnHandle NewRgn (
   void
);
```

Return Value
A handle to the new region.

Discussion
The `NewRgn` function allocates space for a new, variable-size region and initializes it to the empty region defined by the rectangle (0,0,0,0). This is the only function that creates a new region; other functions merely alter the size or shape of existing regions.

To begin defining a region, use the `OpenRgn` (page 2737) function.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

The `NewRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Use the Memory Manager function `MaxMem` to determine whether the memory for the region is valid before using `NewRgn`.

Ensure that the memory for a region is valid before calling this function to manipulate that region if there isn't sufficient memory, the system may crash. Regions are limited to 32 KB in size in basic QuickDraw and 64 KB in color QuickDraw. Before defining a region, you can use the Memory Manager function `MaxMem` to determine whether the memory for the region is valid. You can determine the current size of an existing region by calling the Memory Manager function `GetHandleSize`. When you record drawing operations in an open region, the resulting region description may overflow the 32 KB or 64 KB limit. Should this happen in color QuickDraw, the `QDError` function returns the result code `regionTooBigError`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`QuickdrawAPI.h`

NewScreenBuffer

Creates an offscreen `PixMap` structure and allocates memory for the base address of its pixel image. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDErr NewScreenBuffer (
   const Rect *globalRect,
   Boolean purgeable,
   GDHandle *gdh,
   PixMapHandle *offscreenPixMap
);
```

Parameters

globalRect

The boundary rectangle, in global coordinates, for the offscreen pixel map.

purgeable

A value of `TRUE` to make the memory block for the offscreen pixel map purgeable, or a value of `FALSE` to make it unpurgeable.

gdh

On return, a pointer to the handle to the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle specified in the `globalRect` parameter.

offscreenPixMap

On return, a pointer to a handle to the new offscreen `PixMap` structure.

Return Value
A result code.

Discussion
Applications generally do not need to use `NewScreenBuffer`. The `NewGWorld` (page 2715) function uses the `NewScreenBuffer` function to create and allocate memory for an offscreen pixel image.

Special Considerations

The `NewScreenBuffer` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

NewTempScreenBuffer

Creates an offscreen `PixMap` structure and allocate temporary memory for the base address of its pixel image applications generally don't need to use `NewTempScreenBuffer`. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDErr NewTempScreenBuffer (
   const Rect *globalRect,
   Boolean purgeable,
   GDHandle *gdh,
   PixMapHandle *offscreenPixMap
);
```

Parameters

globalRect

The boundary rectangle, in global coordinates, for the offscreen pixel map.

purgeable

A value of `TRUE` to make the memory block for the offscreen pixel map purgeable, or a value of `FALSE` to make it unpurgeable.

gdh

On return, a pointer to the handle to the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle specified in the `globalRect` parameter.

offscreenPixMap

On return, a pointer to the handle to the new offscreen `PixMap` structure.

Return Value
A result code.

Discussion
The `NewTempScreenBuffer` function performs the same functions as `NewScreenBuffer` except that it creates the base address for the offscreen pixel image in temporary memory. When an application passes it the `useTempMem` flag, the `NewGWorld` function uses `NewTempScreenBuffer` instead of `NewScreenBuffer`.

Your application should not need to use this function.

Special Considerations

The `NewTempScreenBuffer` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

NoPurgePixels

Prevents the Memory Manager from purging the base address for an offscreen pixel image. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void NoPurgePixels (
   PixMapHandle pm
);
```

Parameters

pm

A handle to an offscreen pixel map.

Discussion

The `NoPurgePixels` function marks the base address for an offscreen pixel image as unpurgeable.

To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` (page 2642) function. Then supply this handle for the `pm` parameter of `NoPurgePixels`.

Special Considerations

The `NoPurgePixels` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QDOffscreen.h`

ObscureCursor

Hides the cursor until the next time the user moves the mouse.

```
void ObscureCursor (
   void
);
```

Discussion

Your application normally calls `ObscureCursor` when the user begins to type. Unlike `HideCursor` (page 2668) , `ObscureCursor` has no effect on the cursor level and must not be balanced by a call to `ShowCursor`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

OffscreenVersion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
SInt32 OffscreenVersion (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

OffsetPoly

Moves a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void OffsetPoly (
   PolyHandle poly,
   short dh,
   short dv
);
```

Parameters

poly
A handle to a polygon to move.

dh
The horizontal distance to move the polygon.

dv
The vertical distance to move the polygon.

Discussion
The `OffsetPoly` function moves the polygon whose handle you pass in the `poly` parameter by adding the value you specify in the `dh` parameter to the horizontal coordinates of its points, and by adding the value you specify in the `dv` parameter to the vertical coordinates of all points of its region boundary. If the values of `dh` and `dv` are positive, the movement is to the right and down; if either is negative, the corresponding movement is in the opposite direction. The region retains its size and shape. This does not affect the screen unless you subsequently call a function to draw the region.

`OffsetPoly` is an especially efficient operation, because the data defining a polygon is stored relative to the first point of the polygon and so is not actually changed by `OffsetPoly`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

OffsetRect

Moves a rectangle.

```
void OffsetRect (
    Rect * r,
    short dh,
    short dv
);
```

Parameters

r

A pointer to the rectangle to move.

dh

The horizontal distance to move the rectangle.

dv

The vertical distance to move the rectangle.

Discussion

The `OffsetRect` function moves the rectangle that you specify in the `r` parameter by adding the value you specify in the `dh` parameter to each of its horizontal coordinates and the value you specify in the `dv` parameter to each of its vertical coordinates. If the `dh` and `dv` parameters are positive, the movement is to the right and down; if either is negative, the corresponding movement is in the opposite direction. The rectangle retains its shape and size; it is merely moved on the coordinate plane. The movement does not affect the screen unless you subsequently call a function to draw within the rectangle.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

HID Calibrator

Declared In

`QuickdrawAPI.h`

OffsetRgn

Moves a region.

```
void OffsetRgn (
    RgnHandle rgn,
    short dh,
    short dv
);
```

Parameters

rgn

A handle to the region to move.

dh

The horizontal distance to move the region.

dv

The vertical distance to move the region.

Discussion

The `OffsetRgn` function moves the region whose handle you pass in the `rgn` parameter by adding the value you specify in the `dh` parameter to the horizontal coordinates of all points of its region boundary, and by adding the value you specify in the `dv` parameter to the vertical coordinates of all points of its region boundary. If the values of `dh` and `dv` are positive, the movement is to the right and down; if either is negative, the corresponding movement is in the opposite direction. The region retains its size and shape. This does not affect the screen unless you subsequently call a function to draw the region.

The `OffsetRgn` function is an especially efficient operation, because most of the data defining a region is stored relative to the `rgnBBox` field in its `Region` structure and so is not actually changed by `OffsetRgn`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

OpColor

Sets the maximum color values for the `addPin` and `subPin` arithmetic transfer modes and the weight color for the `blend` arithmetic transfer mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void OpColor (
   const RGBColor *color
);
```

Parameters

color

An `RGBColor` structure that defines a color.

Discussion

Specify the red, green, and blue values in the `RGBColor` structure and specify this structure in the `color` parameter.

If the current graphics port is defined by a `GrafPort` structure, `OpColor` has no effect.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

OpenCPicture

Begins defining a picture in extended version 2 format. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PicHandle OpenCPicture (
   const OpenCPicParams *newHeader
);
```

Parameters

newHeader

An `OpenCPicParams` structure.

Return Value
A handle to a new `Picture` structure. `OpenCPicture` collects your subsequent drawing commands in this structure. Use this handle when referring to the picture in subsequent functions, such as the `DrawPicture` function.

Discussion
When defining a picture, you can use all other QuickDraw drawing functions, with the exception of `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, and `CalcCMask`. You can also use the `PicComment` (page 2748) function to include picture comments in your picture definition.

The `OpenCPicture` function creates a pictures in the extended version 2 format. This format permits your application to specify resolutions when creating images.

Use the `OpenCPicParams` (page 2865) structure you pass in the `newHeader` parameter to specify the horizontal and vertical resolution for the picture, and specify an optimal bounding rectangle for displaying the picture at this resolution. When you later call the `DrawPicture` (page 2610) function to play back the saved picture, supply a destination rectangle, and `DrawPicture` scales the picture so that it is completely aligned with the destination rectangle. To display a picture at a resolution other than that at which it was created, compute an appropriate destination rectangle by scaling its width and height by the following factor:

```
scale factor = destination resolution / source resolution
```

For example, if a picture was created at 300 dpi and you want to display it at 75 dpi, then your application should compute the destination rectangle width and height as 1/4 of those of the picture's bounding rectangle.

The `OpenCPicture` function calls the `HidePen` function, so no drawing occurs on the screen while the picture is open (unless you call the `ShowPen` function just after `OpenCPicture`, or you called `ShowPen` previously without balancing it by a call to `HidePen`).

After defining the picture, close it by using the `ClosePicture` (page 2581) function.

After creating the picture, use the `GetPictInfo` function to gather information about it. The `PictInfo` structure returned by `GetPictInfo` returns the picture's resolution and optimal bounding rectangle.

Special Considerations

When creating a picture, use the `ClosePicture` function to finish it before you open the Printing Manager with the `PrOpen` function. There are two main reasons for this. First, you should allow the printing driver to use as much memory as possible. Second, the Printing Manager creates its own type of graphics port, one that replaces the standard QuickDraw drawing operations stored in the `grafProcs` field of a `CGrafPort` or `GrafPort` structure. To avoid unexpected results when creating a picture, draw into a graphics port created with QuickDraw instead of drawing into a printing port created by the Printing Manager.

After calling `OpenCPicture`, be sure to finish your picture definition by calling `ClosePicture` before you call `OpenCPicture` again. You cannot nest calls to `OpenCPicture`.

Always use the `ClipRect` procedure to specify a clipping region appropriate for your picture before you call `OpenCPicture`. If you do not use `ClipRect` to specify a clipping region, `OpenCPicture` uses the clipping region specified in the current graphics port. If the clipping region is very large (as it is when a graphics port is initialized) and you scale the picture when drawing it, the clipping region can become invalid when `DrawPicture` scales the clipping region—in which case, your picture will not be drawn.

The `OpenCPicture` function may move or purge memory; do not call at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

OpenCursorComponent

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr OpenCursorComponent (
   Component c,
   ComponentInstance *ci
);
```

Return Value

Carbon Porting Notes

This function is not implemented on Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

OpenPicture

Creates a picture which allows you to specify resolutions for your pictures. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PicHandle OpenPicture (
   const Rect *picFrame
);
```

Parameters

picFrame

The bounding rectangle for the picture. The `DrawPicture` function uses this rectangle to scale the picture if you draw it into a destination rectangle of a different size.

Return Value

A handle to a new `Picture` structure. `OpenPicture` collects your subsequent drawing commands in this structure. Use this handle when referring to the picture in subsequent functions, such as the `DrawPicture` function.

Discussion

The `OpenPicture` function, which was created for earlier versions of system software, is described here for completeness. Use the `OpenPicture` function to begin defining a picture.

The `OpenPicture` function calls the `HidePen` function, so no drawing occurs on the screen while the picture is open (unless you call the `ShowPen` function just after `OpenPicture` or you called `ShowPen` previously without balancing it by a call to `HidePen`).

The `OpenPicture` function creates pictures in the version 2 format on computers with Color QuickDraw when the current graphics port is a color graphics port. Pictures created in this format support color drawing operations at 72 dpi. On computers supporting only basic QuickDraw, or when the current graphics port is a basic graphics port, this function creates pictures in version 1 format. Pictures created in version 1 format support only black-and-white drawing operations at 72 dpi.

When defining a picture, you can use all other QuickDraw drawing functions, with the exception of `CopyMask`, `CopyDeepMask`, `SeedFill`, `SeedCFill`, `CalcMask`, and `CalcCMask`. You can also use the `PicComment` (page 2748) function to include picture comments in your picture definition.

After defining the picture, close it by using the `ClosePicture` (page 2581) function. To draw the picture, use the `DrawPicture` (page 2610) function.

Special Considerations

The version 2 and version 1 picture formats support only 72-dpi resolution. The `OpenCPicture` function creates pictures in the extended version 2 format. The extended version 2 format, which is created by the `OpenCPicture` function on all Macintosh computers running System 7, permits your application to specify additional resolutions when creating images.

Version 1 pictures are limited to 32 KB. You can determine the picture size while it is being formed by calling the Memory Manager function `GetHandleSize`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

OpenPoly

Begins defining a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
PolyHandle OpenPoly (
   void
);
```

Return Value
A handle to a new polygon.

Discussion
The `OpenPoly` function starts saving lines for processing as a polygon definition. While a polygon is open, all calls to the `Line` and `LineTo` functions affect the outline of the polygon. Only the line endpoints affect the polygon definition; the pattern mode, pattern, and size do not affect it. The `OpenPoly` function calls the `HidePen` function, so no drawing occurs on the screen while the polygon is open (unless you call the `ShowPen` function just after calling `OpenPoly`, or you called `ShowPen` previously without balancing it by a call to `HidePen`).

A polygon should consist of a sequence of connected lines. The `OpenPoly` function stores the definition for a polygon in a `Polygon` structure.

When a polygon is open, the current graphics port's `polySave` field contains a handle to information related to the polygon definition. If you want to temporarily disable the polygon definition, you can save the current value of this field, set the field to `NULL`, and later restore the saved value to resume the polygon definition.

Even though the onscreen presentation of a polygon is clipped, the definition of a polygon is not; you can define a polygon anywhere on the coordinate plane.

When you are finished calling the line-drawing functions that define your polygon, use the `ClosePoly` (page 2581) function.

Special Considerations

Do not call `OpenPoly` while a region or another polygon is already open.

Polygons are limited to 64 KB. You can determine the polygon size while it is being formed by calling the Memory Manager function `GetHandleSize`.

The `OpenPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

OpenRgn

Begins defining a region. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void OpenRgn (
   void
);
```

Discussion

The `OpenRgn` function allocates temporary memory to start saving lines and framed shapes for processing as a region definition. Call `OpenRgn` only after initializing a region with the `NewRgn` function.

The `NewRgn` function stores the definition for a region in a `Region` structure.

While a region is open, all calls to `Line`, `LineTo`, and the functions that draw framed shapes (except arcs) affect the outline of the region. Only the line endpoints and shape boundaries affect the region definition—the pattern mode, pattern, and size do not affect it.

When you are finished defining the region, call the `CloseRgn` (page 2582) function.

The `OpenRgn` function calls `HidePen`, so no drawing occurs on the screen while the region is open (unless you call `ShowPen` just after `OpenRgn`, or you called `ShowPen` previously without balancing it by a call to `HidePen`). Since the pen hangs below and to the right of the pen location, drawing lines with even the smallest pen changes pixels that lie outside the region you define.

The outline of a region is mathematically defined and infinitely thin, and it separates the bit or pixel image into two groups of pixels: those within the region and those outside it.

A region should consist of one or more closed loops. Each framed shape itself constitutes a loop. Any lines drawn with the `Line` or `LineTo` function should connect with each other or with a framed shape. Even if the onscreen presentation of a region is clipped, the definition of a region is not; you can define a region anywhere on the coordinate plane with complete disregard for the location of various graphics port entities on that plane.

When a region is open, the current graphics port's `rgnSave` field contains a handle to information related to the region definition. If you want to temporarily disable the collection of lines and shapes, you can save the current value of this field, set the field to `NULL`, and later restore the saved value to resume the region definition. Also, calling `SetPort` while a region is being formed discontinues formation of the region until another call to `SetPort` resets the region's original graphics port.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

Regions are limited to 32 KB in size in basic QuickDraw and 64 KB in Color QuickDraw. You can determine the current size of an existing region by calling the Memory Manager function `GetHandleSize`. When you structure drawing operations in an open region, the resulting region description may overflow the 32 KB or 64 KB limit. Should this happen in Color QuickDraw, the `QDError` function returns the result code `regionTooBigError`.

Do not call `OpenRgn` while another region or a polygon is already open. When you are finished constructing the region, use the `CloseRgn` function, which is described next.

The `OpenRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

PackBits

Compresses a data buffer stored in RAM. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PackBits (
   Ptr *srcPtr,
   Ptr *dstPtr,
   short srcBytes
);
```

Parameters

srcPtr
On entry, a pointer to the first byte of a buffer of data to be compressed. On exit, a pointer to the first byte following the bytes compressed.

dstPtr
On entry, a pointer to the first byte in which to store compressed data. On exit, a pointer to the first byte following the compressed data.

srcBytes
The number of bytes of uncompressed data to be compressed. In versions of software prior to version 6.0.2, this number must be 127 or less.

Discussion
You must allocate memory for the destination buffer itself. Allocate enough memory for a worst-case scenario where the destination buffer is 128 bytes long for each block of source data up to 127 bytes. Use the following formula to determine how much space to allocate for the destination buffer:

maxDstBytes := srcBytes + (srcBytes+126) DIV 127;

where `maxDstBytes` stands for the maximum number of destination bytes.

The `PackBits` algorithm is most effective on data buffers in which there are likely to be series of bytes containing the same value. For example, resources of many formats often contain many consecutive zeros. If you have a data buffer in which there are only likely to be a series of words or long words containing the same values, `PackBits` is unlikely to be effective.

Special Considerations

Because your application must allocate memory for the source and destination buffers, `PackBits` does not move relocatable blocks. Thus, you can call it at interrupt time.

Because `PackBits` changes the values of the `srcPtr` and `dstPtr` parameters, you should pass to `PackBits` only copies of the pointers to the source and destination buffers. This allows you to access the beginning of the source and destination buffers after `PackBits` returns. Also, if the source or destination buffer is stored in an unlocked, relocatable block, this technique prevents `PackBits` from changing the value of a master pointer, which would make the original handle invalid.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

PaintArc

Paints a wedge of the oval that fits inside a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PaintArc (
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

r
The rectangle that defines an oval's boundaries.

startAngle
The angle indicating the start of the arc.

arcAngle
The angle indicating the arc's extent.

Discussion
Using the pen pattern and pattern mode of the current graphics port, the `PaintArc` function draws a wedge of the oval bounded by the rectangle that you specify in the `r` parameter. As in the `FrameArc` function, use the `startAngle` and `arcAngle` parameters to define the arc of the wedge.

The pen location does not change.

Use `FillArc` (page 2619) , to draw a wedge with a pattern different from that specified in the `pnPat` field of the current graphics port.

Special Considerations

The `PaintArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

PaintOval

Paints an oval with the graphics pen's pattern and pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PaintOval (
   const Rect *r
);
```

Parameters

r

The rectangle that defines the oval's boundary.

Discussion

Using the pen pattern and pattern mode for the current graphics port, the `PaintOval` function draws the interior of an oval just inside the bounding rectangle that you specify in the `r` parameter. The pen location does not change.

Use `FillOval` (page 2624) to draw the interior of an oval with a pen pattern different from that for the current graphics port.

Special Considerations

The `PaintOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PaintPoly

Paints a polygon with the graphics pen's pattern and pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PaintPoly (
   PolyHandle poly
);
```

Parameters

poly

A handle to the polygon to paint. The `OpenPoly` (page 2737) function returns this handle when you first create the polygon.

Discussion

Using the pen pattern and pattern mode for the current graphics port, the `PaintPoly` function draws the interior of a polygon whose handle you pass in the `poly` parameter. The pen location does not change.

This function temporarily converts the polygon into a region to perform their operations. The amount of memory required for this temporary region may be far greater than the amount required by the polygon alone.

You can estimate the size of this region by scaling down the polygon with the `MapPoly` (page 2707) , converting the polygon into a region, checking the region's size with the Memory Manager function `GetHandleSize`, and multiplying that value by the factor by which you scaled the polygon.

The result of this graphics operation is undefined whenever any horizontal or vertical line drawn through the polygon would intersect the polygon's outline more than 50 times.

Use the `FillPoly` (page 2625) function to draw the interior of a polygon with a pattern different from that specified in the `pnPat` field of the current graphics port.

Special Considerations

Do not create a height or width for the polygon greater than 32,767 pixels, or `PaintPoly` will crash.

The `PaintPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PaintRect

Paints a rectangle with the graphics pen's pattern and pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PaintRect (
   const Rect *r
);
```

Parameters

r

The rectangle to paint.

Discussion

The `PaintRect` function draws the interior of the rectangle that you specify in the `r` parameter with the pen pattern for the current graphics port, according to the pattern mode for the current graphics port. The pen location does not change.

Use the `FillRect` (page 2626) to draw the interior of a rectangle with a pen pattern different from that for the current graphics port.

Special Considerations

The `PaintRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
HideMenuBar
Simple DrawSprocket

Declared In
`QuickdrawAPI.h`

PaintRgn

Paints a region with the graphics pen's pattern and pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PaintRgn (
    RgnHandle rgn
);
```

Parameters

rgn

A handle to the region to paint.

Discussion
Using the pen pattern and pattern mode for the current graphics port, the `PaintRgn` function draws the interior of the region whose handle you pass in the `rgn` parameter. The pen location does not change.

This function depends on the local coordinate system of the current graphics port. If you draw a region in a graphics port different from the one in which you defined the region, it may not appear in the proper position in the graphics port.

If any horizontal or vertical line drawn through the region would intersect the region's outline more than 50 times, the results of this graphics operation are undefined.

Use `FillRgn` (page 2626) to draw the interior of a region with a pen pattern different from that for the current graphics port.

Special Considerations

The `PaintRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

PaintRoundRect

Paints a rounded rectangle with the graphics pen's pattern and pattern mode. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PaintRoundRect (
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

r

The rectangle that defines the rounded rectangle's boundaries.

ovalWidth

The width of the oval defining the rounded corner.

ovalHeight

The height of the oval defining the rounded corner.

Discussion

Using the pattern and pattern mode of the graphics pen for the current graphics port, the `PaintRoundRect` function draws the interior of the rounded rectangle bounded by the rectangle that you specify in the `r` parameter. Use the `ovalWidth` and `ovalHeight` parameters to specify the diameters of curvature for the corners of the rounded rectangle.

The pen location does not change.

Use `FillRoundRect` (page 2627) to draw the interior of a rounded rectangle with a pen pattern different from that for the current graphics port.

Special Considerations

The `PaintRoundRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PenMode

Sets the pattern mode of the graphics pen in the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PenMode (
   short mode
);
```

Parameters

mode

The pattern mode. See "Source, Pattern, and Arithmetic Transfer Mode Constants" (page 2898).

Discussion

Using the pattern mode you specify in the `mode` parameter, the `PenMode` function sets the manner in which the pattern of the graphics pen is transferred onto the bitmap (or pixel map) when you draw lines or shapes in the current graphics port.

If you specify a source mode (such as one used with the `CopyBits` function) instead of a pattern mode, no drawing is performed.

The current pattern mode is stored in the `pnMode` field of the current graphics port. The initial pattern mode value is `patCopy`, in which the pen pattern is copied directly to the bitmap.

To use highlighting, add the `hilite` constant or its value to the source or pattern mode:

With highlighting, QuickDraw replaces the background color with the highlight color when your application draws or copies images between graphics ports. This has the visual effect of using a highlighting pen to select the object. (The global variable `HiliteRGB` is read from parameter RAM when the machine starts. Basic graphics ports use the color stored in the `HiliteRGB` global variable as the highlight color. Color graphics ports default to the `HiliteRGB` global variable, but can be overridden by the `HiliteColor` function.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Special Considerations

When your application draws with a pixel pattern, Color QuickDraw ignores the pattern mode and simply transfers the pattern directly to the pixel map without regard to the foreground and background colors.

The results of inverting a pixel are predictable only with direct pixels or 1-bit pixel maps. For indexed pixels, Color QuickDraw performs the inversion on the pixel indexes, which means the results depend entirely on the contents of the color table. The eight colors used in basic QuickDraw are stored in a color table represented by the global variable `QDColors`. To display those eight basic QuickDraw colors on an indexed device, Color QuickDraw uses the Color Manager to obtain indexes to the colors in the CLUT that best map to the colors in the `QDColors` color table. Because the index, not the color value, is inverted, the results are unpredictable.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PenNormal

Sets the size, pattern, and pattern mode of the graphics pen in the current graphics port to their initial values. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PenNormal (
   void
);
```

Discussion

The `PenNormal` function restores the size, pattern, and pattern mode of the graphics pen in the current graphics port to their initial values: a size of 1 pixel by 1 pixel, a pattern mode of `patCopy`, and a pattern of `black`. The pen location does not change.

Special Considerations

The `PenNormal` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PenPat

Sets the bit pattern to be used by the graphics pen in the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PenPat (
   const Pattern *pat
);
```

Parameters

pat

A bit pattern, as defined by a `Pattern` structure.

Discussion

The `PenPat` function sets the graphics pen to use the bit pattern defined in the `Pattern` (page 2866) structure that you specify in the `pat` parameter. (The standard patterns `white`, `black`, `gray`, `ltGray`, and `dkGray` are predefined; the initial bit pattern for the pen is `black`.) This pattern is stored in the `pnPat` field of a `GrafPort` structure. The QuickDraw painting functions (such as `PaintRect`) also use the pen's pattern when drawing a shape.

The `PenPat` function also sets a bit pattern for the graphics pen in a color graphics port. The `PenPat` function creates a handle, of type `PixPatHandle`, for the bit pattern and stores this handle in the `pnPixPat` field of the `CGrafPort` structure. This pattern always uses the graphics port's current foreground and background colors.

To define your own patterns, you typically create pattern, 'PAT', or pattern list, 'PAT#', resources.

Special Considerations

The `PenPat` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PenPixPat

Sets the pixel pattern used by the graphics pen in the current color graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PenPixPat (
   PixPatHandle pp
);
```

Parameters

pp

A handle to the pixel pattern to use as the pen pattern.

Discussion

The `PenPixPat` function is similar to the basic QuickDraw function `PenPat`, except that you pass `PenPixPat` a handle to a multicolored pixel pattern rather than a bit pattern.

The `PenPixPat` function stores the handle to the pixel pattern in the `pnPixPat` field of the `CGrafPort` structure, therefore, you should not dispose of this handle since QuickDraw removes all references to your pattern from an existing graphics port when you dispose of it.

If you use `PenPixPat` to set a pixel pattern in a basic graphics port, the data in the `pat1Data` field of the `PixPat` (page 2871) structure is placed into the `pnPat` field of the `GrafPort` structure.

To define your own pixel pattern, you can create a pixel pattern resource, which is described in `'ppat'`, or you can use the `NewPixPat` (page 2720) function. To set the pen to use a bit pattern, you can also use the QuickDraw function `PenPat`.

Special Considerations

The `PenPixPat` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PenSize

Sets the dimensions of the graphics pen in the current graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PenSize (
   short width,
   short height
);
```

Parameters

width

The pen width, as an integer from 0 to 32,767. If you set the value to 0, the pen does not draw. Values less than 0 are undefined.

height

The pen height, as an integer from 0 to 32,767. If you set the value to 0, the pen does not draw. Values less than 0 are undefined.

Discussion

The `PenSize` function sets the width that you specify in the `width` parameter and the height that you specify in the `height` parameter for the graphics pen in the current graphics port. All subsequent calls to the `Line` and `LineTo` functions and to the functions that draw framed shapes in the current graphics port use the new pen dimensions.

You can get the current pen dimensions from the `pnSize` field of the current graphics port, where the width and height are stored as a `Point` structure.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PicComment

Inserts a picture comment into a picture that you are defining or into your printing code. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PicComment (
   short kind,
   short dataSize,
   Handle dataHandle
);
```

Parameters

kind

The type of comment.

dataSize

Size of any additional data passed in the `dataHandle` parameter. If no additional data is used, specify 0 in this parameter.

dataHandle

A handle to additional data, if used. If no additional data is used, specify `NULL` in this parameter.

Discussion

When used after your application begins creating a picture with the `OpenCPicture` (or `OpenPicture`) function, the `PicComment` function inserts the specified comment into the `Picture` structure. When sent to a printer driver after your application uses the `PrOpenPage` function, `PicComment` passes the data or commands in the specified comment directly to the printer.

Picture comments contain data or commands for special processing by output devices, such as printers.

Usually printer drivers process picture comments, but applications can also do so. For your application to process picture comments, it must replace the `StdComment` function pointed to by the `commentProc` field of the `CQDProcs` or `QDProcs` structure, which in turn is pointed to by the `grafProcs` field of a `CGrafPort` or `GrafPort` structure. The default `StdComment` function provided by QuickDraw does no comment processing whatsoever. You can use the `SetStdCProcs` function to assist you in changing the `CQDProcs` structure, and you can use the `SetStdProcs` function to assist you in changing the `QDProcs` structure.

If you create and process your own picture comments, you should define comments so that they contain information that identifies your application (to avoid using the same comments as those used by Apple or by other third-party products). You should define a comment as an `ApplicationComment` comment type with a `kind` value of 100. The first 4 bytes of the data for the comment should specify your application's signature. You can use the next 2 bytes to identify the type of comment—that is, to specify a `kind` value to your own application.

Suppose your application signature were `'WAVE'`, and you wanted to use the value 128 to identify a `kind` value to your own application. You would supply values to the `kind` and `data` parameters to `PicComment` as follows:

`kind` = 100; `data` = `'WAVE'` [4 bytes] + 128 [2 bytes] + additional data [n bytes]

Your application can then parse the first 6 bytes of the comment to determine whether and how to process the rest of the data in the comment. It is up to you to publish information about your comments if you wish them to be understood and used by other applications.

Special Considerations

These former picture comments are now obsolete: `SetGrayLevel`, `ResourcePS`, `PostScriptFile`, and `TextIsPostScript`.

The `PicComment` function may move or purge memory.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

PixMap32Bit

Determines whether a pixel map requires 32-bit addressing mode for access to its pixel image. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean PixMap32Bit (
   PixMapHandle pmHandle
);
```

Parameters

pmHandle

A handle to an offscreen pixel map.

Return Value

`TRUE` if a pixel map requires 32-bit addressing mode for access to its pixel image. If your application is in 24-bit mode, you must change to 32-bit mode.

Discussion
To get a handle to an offscreen pixel map, first use the `GetGWorldPixMap` (page 2642) function. Then supply this handle for the `pm` parameter of `PixMap32Bit`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

PixPatChanged

Notifies QuickDraw that the content of a `PixPat` structure, including its `PixMap` structure or the image in its `patData` field, has been modified. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PixPatChanged (
   PixPatHandle ppat
);
```

Parameters

ppat
A handle to the changed pixel pattern.

Discussion
The `PixPatChanged` function sets the `patXValid` field of the `PixPat` structure specified in the `ppat` parameter to –1 and notifies QuickDraw of the change.

If your application changes the `pmTable` field of a pixel pattern's `PixMap` structure, it should call `PixPatChanged`. However, if your application changes the content of the color table referenced by the `PixMap` structure's `pmTable` field, it should call both the `PixPatChanged` and the `CTabChanged` functions.

Your application should never need to directly modify a `PixPat` structure and use the `PixPatChanged` function; instead, your application should use the QuickDraw functions for manipulating the values in a `PixPat` structure.

Special Considerations

The `PixPatChanged` function may move or purge memory in the application heap; do not call the `PixPatChanged` function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QDOffscreen.h`

PortChanged

Notifies QuickDraw that the content of a `GrafPort` structure or `CGrafPort` structure, including any of the data structures specified by handles within the structure, has been modified. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PortChanged (
   GrafPtr port
);
```

Parameters

port

A pointer to the `GrafPort` structure that you have changed.

Discussion

If your application has changed a `CGrafPort` structure, it must coerce the `CGrafPtr` so it will point to a `GrafPtr` before passing the pointer in the `port` parameter.

You generally should not directly change any of the `PixPat` structures specified in a `CGrafPort` structure, but instead use the `PenPixPat` and `BackPixPat` functions. However, if your application does change the content of a `PixPat` structure, it should call the `PixPatChanged` function and the `PortChanged` function.

If your application changes the `pmTable` field of the `PixMap` structure specified in the graphics port, your application should call `PortChanged`. If your application changes the content of the `ColorTable` structure referenced by the `pmTable` field, it should call `CTabChanged` also.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QDOffscreen.h`

PortSize

Changes the size of the port rectangle of the current graphics port (basic or color). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void PortSize (
   short width,
   short height
);
```

Parameters

width

The width of the reset port rectangle.

height

The height of the reset port rectangle.

Discussion
The `PortSize` function is normally called only by the Window Manager. The `PortSize` function changes the size of the current graphics port's port rectangle. The upper-left corner of the port rectangle remains at its same location the width and height of the port rectangle are set to the given `width` and `height`. In other words, `PortSize` moves the lower-right corner of the port rectangle to a position relative to the upper-left corner.

The `PortSize` function doesn't change the clipping or visible region of the graphics port, nor does it affect the local coordinate system of the graphics port it changes only the width and height of the port rectangle. Remember that all drawing occurs only in the intersection of the boundary rectangle and the port rectangle, after being cropped to the visible region and the clipping region.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ProtectEntry

Adds protection to or removes protection from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ProtectEntry (
   short index,
   Boolean protect
);
```

Parameters
index
The index to the entry whose protection is to be changed.
protect
A Boolean value: specify `true` to protect the entry, `false` to remove protection.

Discussion
A protected entry can not be changed by other applications. `ProtectEntry` returns a protection error in `QDErr` if you attempt to protect an already protected entry. However, it can remove protection from any entry, even an already unprotected one.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

Pt2Rect

Determines the smallest rectangle that encloses two given points.

```
void Pt2Rect (
   Point pt1,
   Point pt2,
   Rect *dstRect
);
```

Parameters

pt1

The first of two points to enclose.

pt2

The second of two points to enclose.

dstRect

On return, a pointer to the smallest rectangle that can enclose them.

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawAPI.h`

PtInRect

Determines whether a pixel below is enclosed in a rectangle.

```
Boolean PtInRect (
    Point pt,
    const Rect * r
);
```

Parameters

pt

The point to test.

r

The rectangle to test.

Return Value

`TRUE` if the pixel below and to the right of the point specified in the `pt` parameter is enclosed in the rectangle specified in the `r` parameter. `FALSE` if it is not.

Discussion
If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawAPI.h`

PtInRgn

Determines whether a pixel is within a region.

```
Boolean PtInRgn (
   Point pt,
   RgnHandle rgn
);
```

Parameters

pt
The point whose pixel is to be checked.

rgn
A handle to the region to test.

Return Value
`TRUE` if the pixel below and to the right of the point specified in the `pt` parameter is within the region whose handle is specified in the `rgn` parameter. `FALSE` if it is not.

Discussion
If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

PtToAngle

Calculates an angle between a vertical line pointing straight up from the center of a rectangle and a line from the center to a given point.

```
void PtToAngle (
   const Rect *r,
   Point pt,
   short *angle
);
```

Parameters

r

The rectangle to examine.

pt

The point to which an angle is to be calculated.

angle

On return, a pointer to the resulting angle.

The result returned in the `angle` parameter is specified in degrees from 0 to 359, measured clockwise from 12 o'clock, with 90 at 3 o'clock, 180 at 6 o'clock, and 270 at 9 o'clock. Other angles are measured relative to the rectangle. If the line to the given point goes through the upper-right corner of the rectangle, the angle returned is 45, even if the rectangle is not square if it goes through the lower-right corner, the angle is 135, and so on.

The angle returned can be used as input to one of the functions that manipulate arcs and wedges, in "Drawing Arcs and Wedges".

Discussion

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawAPI.h`

QDAddRectToDirtyRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDAddRectToDirtyRegion (
   CGrafPtr inPort,
   const Rect *inBounds
);
```

Return Value

Availability

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDAddRegionToDirtyRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDAddRegionToDirtyRegion (
   CGrafPtr inPort,
   RgnHandle inRegion
);
```

Return Value

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDBeginCGContext

Returns a Quartz 2D drawing environment associated with a graphics port.

```
OSStatus QDBeginCGContext (
   CGrafPtr inPort,
   CGContextRef *outContext
);
```

Parameters

port

A color graphics port in which to draw. Offscreen graphics worlds with pixel depths of 1, 2, 4, and 8 are not supported. When using Quartz 2D to draw in a offscreen graphics world, alpha information is always ignored.

contextPtr

A pointer to your storage for a Quartz context. Upon completion, `contextPtr` points to a context associated with the port. The context matches the port's pixel depth, width, and height. Otherwise the context is in a default state and does not necessarily match other port attributes such as foreground color, background color, or clip region.

You should not retain or release the context. When you are finished using the context, you should call `QDEndCGContext` (page 2758).

Return Value
A result code. If `noErr`, the context was successfully initiated.

Discussion
Applications running in Mac OS X can use Quartz 2D to draw in a QuickDraw graphics port. When you call this function, you obtain a Quartz context that's associated with the specified port. To improve performance, contexts returned by this function are cached and reused during subsequent calls whenever possible.

Each block of Quartz 2D drawing code in your application should be surrounded by calls to this function and `QDEndCGContext` (page 2758). Nested calls to this function for the same graphics port are not permitted—that is, for a given port you should not call this function more than once without an intervening call to `QDEndCGContext` (page 2758).

While the Quartz context is in use, all Quickdraw imaging operations in the associated graphics port are disabled. This is done because the operations would fail during printing.

For information about how to use a Quartz context, see *Quartz 2D Programming Guide*.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`QuickdrawAPI.h`

QDDisplayWaitCursor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void QDDisplayWaitCursor (
   Boolean forceWaitCursor
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDDisposeRegionBits

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDDisposeRegionBits (
   QDRegionBitsRef regionBits
);
```

Return Value

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDDone

Determines whether QuickDraw has completed drawing in a given graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean QDDone (
   GrafPtr port
);
```

Parameters

port

The GrafPort structure for a graphics port in which your application has begun drawing; if you pass NULL, QDDone tests all open graphics ports.

Return Value
TRUE if all drawing operations have finished in the graphics port specified in the port parameter, FALSE if any remain to be executed. If you pass NULL in the port parameter, then QDDone returns TRUE only if drawing operations have completed in all ports.

Discussion
The QDDone function may be useful if a graphics accelerator is present and operating asynchronously. You can use it to ensure that all drawing is done before issuing new drawing commands, and to avoid the possibility that the new drawing operations might be overlaid by previously issued but unexecuted operations.

Special Considerations

If a graphics port draws a clock or some other continuously operating drawing process, QDDone may never return TRUE.

To determine whether all drawing in a color graphics port has completed, you must coerce its CGrafPort structure to a GrafPort structure, which you pass in the port parameter.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QDOffscreen.h

QDEndCGContext

Terminates a Quartz 2D drawing environment associated with a graphics port.

```
OSStatus QDEndCGContext (
   CGrafPtr inPort,
   CGContextRef *inoutContext
);
```

Parameters

port

A graphics port specified in a preceding call to `QDBeginCGContext` (page 2756).

contextPtr

A pointer to the context obtained in the preceding call to `QDBeginCGContext` (page 2756) for the port. Upon completion, the storage pointed to by `contextPtr` is set to `NULL`.

Return Value
A result code. If `noErr`, the context is terminated.

Discussion
After you finish using Quartz 2D to draw in a graphics port, you should call this function to terminate the context. For more information, see `QDBeginCGContext` (page 2756).

Before calling this function, you should do one of the following:

- Call `CGContextSynchronize` (page 136) to mark the affected areas of the port for update.
- Call `CGContextFlush` (page 95) to immediately update the destination device.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`QuickdrawAPI.h`

QDError

Obtains a result code from the last applicable QuickDraw function that you called. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
short QDError (
   void
);
```

Return Value
The error result. On a system with only basic QuickDraw, `QDError` always returns `noErr`.

Discussion
The `QDError` function is helpful in determining whether insufficient memory caused a drawing operation - particularly those involving regions, polygons, pictures, and images copied with `CopyBits` - to fail.

Basic QuickDraw uses stack space for work buffers. For complex operations such as depth conversion, dithering, and image resizing, stack space may not be sufficient. QuickDraw attempts to get temporary memory from other parts of the system. If that is still not enough, `QDError` returns the `nsStackErr` error. If your application receives this result, reduce the memory required by the operation.

When you structure drawing operations in an open region, the resulting region description may overflow the 64 KB limit. In this case, `QDError` returns `regionTooBigError`. Since the resulting region is potentially corrupt, the `CloseRgn` function returns an empty region if it detects `QDError` has returned `regionTooBigError`. A similar error, `rgnTooBigErr`, occurs when using the `BitMapToRegion` function to convert a bitmap to a region.

The `BitMapToRegion` function also generates the `pixmapTooDeepErr` error if a `PixMap` structure is supplied that is greater than 1 bit per pixel. You may be able to recover from this problem by coercing your `PixMap` structure into a 1-bit `PixMap` structure and calling the `BitMapToRegion` function again.

Special Considerations

The `QDError` function does not report errors returned by basic QuickDraw.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

QDFlushPortBuffer

Calls the Quartz compositor to flush all new drawing in a Carbon window to the display. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void QDFlushPortBuffer (
   CGrafPtr port,
   RgnHandle region
);
```

Parameters

port

A window port. If the port has no back buffer, or if the port is an offscreen or printing port, this function does nothing.

region

An update region. Under normal conditions, you should pass `NULL` to avoid the overhead of additional region operations.

Discussion

In Mac OS X, drawing in a window port updates a back buffer associated with the window. Updates to this buffer are accumulated in a list called the dirty region.

The back buffer is automatically flushed to the display each time through the event loop. When the event loop does not get control soon enough—for example, during an animation sequence—you can call this function to flush the port buffer to the device immediately.

When you call this function, there are several different execution paths:

1. If the `region` parameter is `NULL`, the dirty region is flushed—along with any Quartz 2D drawing operations marked for update by calls to `CGContextSynchronize` (page 136)—and the dirty region is set to empty.

2. If the `region` parameter specifies an update region, the intersection of the dirty region and the update region is flushed—along with any Quartz 2D drawing operations marked for update by calls to `CGContextSynchronize` (page 136)—and the flushed region is subtracted from the dirty region.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`QuickdrawAPI.h`

QDGetCGDirectDisplayID

Returns the Quartz display ID that corresponds to a QuickDraw graphics device.

```
CGDirectDisplayID QDGetCGDirectDisplayID (
   GDHandle inGDevice
);
```

Parameters

`inGDevice`
A QuickDraw graphics device.

Return Value
A Quartz display ID, or `NULL` if the `inGDevice` parameter does not represent a display. For information about using a display ID, see *Quartz Display Services Reference*.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDGetCursorData

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDGetCursorData (
   Boolean contextCursor,
   PixMapHandle *crsrData,
   Point *hotSpot
);
```

Return Value

Availability
Available in Mac OS X v10.3 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDGetDirtyRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDGetDirtyRegion (
   CGrafPtr port,
   RgnHandle rgn
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDGetPatternOrigin

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void QDGetPatternOrigin (
   Point *origin
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDGetPictureBounds

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Rect * QDGetPictureBounds (
   PicHandle picH,
   Rect *outRect
);
```

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDGlobalToLocalPoint

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Point * QDGlobalToLocalPoint (
   CGrafPtr port,
   Point *point
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`QuickdrawAPI.h`

QDGlobalToLocalRect

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Rect * QDGlobalToLocalRect (
   CGrafPtr port,
   Rect *bounds
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDGlobalToLocalRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RgnHandle QDGlobalToLocalRegion (
   CGrafPtr port,
   RgnHandle region
);
```

Return Value

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDIsNamedPixMapCursorRegistered

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean QDIsNamedPixMapCursorRegistered (
   const char name[128]
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDIsPortBufferDirty

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean QDIsPortBufferDirty (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDIsPortBuffered

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean QDIsPortBuffered (
   CGrafPtr port
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDLocalToGlobalPoint

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Point * QDLocalToGlobalPoint (
   CGrafPtr port,
   Point *point
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDLocalToGlobalRect

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Rect * QDLocalToGlobalRect (
   CGrafPtr port,
   Rect *bounds
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
QuickdrawAPI.h

QDLocalToGlobalRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
RgnHandle QDLocalToGlobalRegion (
   CGrafPtr port,
   RgnHandle region
);
```

Return Value

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDPictCreateWithProvider

Creates a QDPict picture, using QuickDraw picture data supplied with a Quartz data provider.

```
QDPictRef QDPictCreateWithProvider (
   CGDataProviderRef provider
);
```

Parameters

provider

A Quartz data provider that supplies QuickDraw picture data. The picture data must begin at either the first byte or the 513th byte in the data provider. The picture bounds must not be an empty rectangle.

QuickDraw retains the data provider you pass in, and you may safely release it after this function returns.

Return Value
A new QDPict picture, or NULL if the picture data is not valid. The initial retain count is 1. After you finish using the picture, you should release it by calling QDPictRelease (page 2769).

Discussion
This function creates a QDPict picture that you can draw in a Quartz context. For general information about QDPict pictures, see QDPictRef (page 2877).

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
QDPictToCGContext.h

QDPictCreateWithURL

Creates a QDPict picture, using QuickDraw picture data specified with a Core Foundation URL.

```
QDPictRef QDPictCreateWithURL (
   CFURLRef url
);
```

Parameters

url

A Core Foundation URL that specifies a PICT file containing the QuickDraw picture data. The picture header data must begin at either the first byte or the 513th byte in the PICT file. The picture bounds must not be an empty rectangle.

Return Value

A new QDPict picture, or `NULL` if the picture data is not valid. The initial retain count is 1. After you finish using the picture, you should release it by calling `QDPictRelease` (page 2769).

Discussion

This function creates a QDPict picture that you can draw in a Quartz context. For general information about QDPict pictures, see `QDPictRef` (page 2877).

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`QDPictToCGContext.h`

QDPictDrawToCGContext

Draws a QuickDraw picture in a Quartz context.

```
OSStatus QDPictDrawToCGContext (
   CGContextRef ctx,
   CGRect rect,
   QDPictRef pictRef
);
```

Parameters

context

The Quartz context in which to draw.

rect

The rectangular area in which to draw the picture. You should specify the origin and size of this rectangle in user space units. The origin is the lower left corner of the picture when drawn. If necessary, the picture is scaled to fit inside this rectangle. To get unscaled results, you should pass the rectangle returned by `QDPictGetBounds` (page 2768). For additional information about scaling, see the discussion below.

picture

A QDPict picture.

Return Value

A result code. A non-zero result indicates that the picture was not successfully drawn.

Discussion
This function converts the picture data in a QDPict picture into an equivalent sequence of Quartz 2D graphics operations. Conceptually this is the same processing path taken when an application running in Mac OS X draws into a QuickDraw printing port.

When drawing a QDPict picture in a Quartz context, there are two ways to change the horizontal or vertical scale of the picture:

- Construct the drawing rectangle (see the `rect` parameter) by applying the change of scale to the bounds rectangle returned by `QDPictGetBounds` (page 2768). In this case, QuickDraw scales all the graphic elements in the picture except for patterns—the same behavior as `DrawPicture` (page 2610).
- Prior to calling `QDPictDrawToCGContext`, apply the change of scale to the current transformation matrix in the Quartz context—for example, by calling `CGContextScaleCTM` (page 105). In this case, QuickDraw scales the entire picture including patterns.

In a bitmap-based context, the picture is rendered into the bitmap. In a PDF-based context, the picture is converted into a PDF content stream. If the picture uses transfer modes such as `srcXor` that do not have an analog in Quartz 2D, the PDF representation may not match the original exactly.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`QDPictToCGContext.h`

QDPictGetBounds

Returns the intended location and size of a QDPict picture.

```
CGRect QDPictGetBounds (
   QDPictRef pictRef
);
```

Parameters
picture
A QDPict picture.

Return Value
A Quartz rectangle that represents the intended location and size of the picture. The rectangle is in default user space with one unit = 1/72 inch, and the origin is the lower-left corner of the picture.

Discussion
If the native resolution in the picture data is not 72 pixels per inch, the bounding rectangle returned by this function is scaled as follows:

```
width = width in pixels * 72 / horizontal resolution
height = height in pixels * 72 / vertical resolution
```

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
QDPictToCGContext.h

QDPictGetResolution

Returns the horizontal and vertical resolution of a QDPict picture.

```
void QDPictGetResolution (
   QDPictRef pictRef,
   float *xRes,
   float *yRes
);
```

Parameters

picture
A QDPict picture.

xRes
A pointer to your storage for a return value. Upon completion, the value is the picture's horizontal resolution in pixels per inch.

yRes
A pointer to your storage for a return value. Upon completion, the value is the picture's vertical resolution in pixels per inch.

Discussion
This function returns resolution data that you can use—together with the rectangle returned by `QDPictGetBounds` (page 2768)—to compute the picture's size in pixels.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
QDPictToCGContext.h

QDPictRelease

Releases a QDPict picture.

```
void QDPictRelease (
   QDPictRef pictRef
);
```

Parameters

picture
A QDPict picture which you created or retained.

Discussion
After you finish using a QDPict picture that you created or retained, you should call this function to release the picture. If the picture's retain count becomes 0, this function frees the picture and any associated resources such as the picture's data provider.

Availability
Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In
QDPictToCGContext.h

QDPictRetain

Retains a QDPict picture.

```
QDPictRef QDPictRetain (
   QDPictRef pictRef
);
```

Parameters

picture
A QDPict picture.

Return Value
The retained picture.

Discussion
You should call this function when you obtain a QDPict picture that you did not create and you want to retain the picture for later use. When you no longer need the retained picture, you should call `QDPictRelease` (page 2769) to release it.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
QDPictToCGContext.h

QDRegionToRects

```
OSStatus QDRegionToRects (
   RgnHandle rgn,
   QDRegionParseDirection dir,
   RegionToRectsUPP proc,
   void *userData
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawAPI.h

QDRegisterNamedPixMapCursor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDRegisterNamedPixMapCursor (
   PixMapHandle crsrData,
   PixMapHandle crsrMask,
   Point hotSpot,
   const char name[128]
);
```

Return Value

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDRestoreRegionBits

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDRestoreRegionBits (
   RgnHandle region,
   QDRegionBitsRef regionBits
);
```

Return Value

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDSaveRegionBits

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
QDRegionBitsRef QDSaveRegionBits (
   RgnHandle region
);
```

Return Value

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDSetCursorScale

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDSetCursorScale (
   float scale
);
```

Return Value

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDSetDirtyRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDSetDirtyRegion (
   CGrafPtr port,
   RgnHandle rgn
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDSetNamedPixMapCursor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDSetNamedPixMapCursor (
   const char name[128]
);
```

Return Value

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDSetPatternOrigin

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void QDSetPatternOrigin (
   Point origin
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

QDSwapPort

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean QDSwapPort (
   CGrafPtr inNewPort,
   CGrafPtr *outOldPort
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
QuickdrawAPI.h

QDSwapPortTextFlags

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
UInt32 QDSwapPortTextFlags (
   CGrafPtr port,
   UInt32 newFlags
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDSwapTextFlags

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
UInt32 QDSwapTextFlags (
   UInt32 newFlags
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

QDUnregisterNamedPixMapCursor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus QDUnregisterNamedPixMapCursor (
   const char name[128]
);
```

Return Value

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

Random

Obtains a pseudorandom integer. (Deprecated in Mac OS X v10.4. Use the Standard C Library `random(3)` function instead.)

```
short Random (
   void
);
```

Return Value
A pseudorandom integer, uniformly distributed in the range -32767 to 32767.

Discussion
The value `Random` returns depends solely on the global variable `randSeed`, which the QuickDraw `InitGraf` function initializes to 1. Each time the `Random` function executes, it uses a numerical algorithm to change the value of `randSeed` to prevent it from returning the same value each time it is called.

To prevent your application from generating the same sequence of pseudo-random numbers each time it is executed, initialize the `randSeed` global variable, when your application starts up, to a volatile long word variable such as the current date and time. If you would like to generate the same sequence of pseudo-random numbers twice, on the other hand, simply set `randSeed` to the same value before calling `Random` for each sequence.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

RealColor

Determines whether a given `RGBColor` data structure exists in the current device's color table. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean RealColor (
   const RGBColor *color
);
```

Parameters
color

The `RGBColor` data structure to be tested.

Discussion
The `RealColor` function determines whether the color is available in the current `GDevice` data structure's CLUT, basing its search on the current resolution of the inverse table. For example, if the current value of the `iTabRes` field is 4, `RealColor` returns `true` if there exists a color that exactly matches the top 4 bits of red, green, and blue. (See the `iTabRes` field of the inverse table, `ITab` (page 2863).)

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

RectInRgn

Determines whether a rectangle intersects a region.

```
Boolean RectInRgn (
   const Rect *r,
   RgnHandle rgn
);
```

Parameters

r

The rectangle to check for intersection.

rgn

A handle to the region to check.

Return Value
TRUE if the rectangle specified in the r parameter intersects the region whose handle is specified in the rgn parameter. The RectInRgn function returns TRUE if the intersection encloses at least 1 bit or FALSE if it does not.

Discussion
If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the SetPort function to change to the graphics port containing the points or rectangles, using the LocalGlobal function to convert their locations to global coordinates, using SetPort to return to your starting graphics port, and then using the GlobalToLocal function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

The RectInRgn function sometimes returns TRUE when the rectangle merely intersects the region's bounding rectangle. If you need to know exactly whether a given rectangle intersects the actual region, use RectRgn (page 2776) to set the rectangle to a region, and call SectRgn (page 2786) to see whether the two regions intersect. If the result of SectRgn is an empty region, then the rectangle does not intersect the region.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

RectRgn

Changes the structure of an existing region to that of a rectangle.

```
void RectRgn (
   RgnHandle rgn,
   const Rect *r
);
```

Parameters

rgn

A handle to the region to restructure as a rectangle.

r

The rectangle structure to use.

Discussion

The `RectRgn` function destroys the previous structure of the `SetRectRgn` function, and it then sets the new structure to a rectangle that you specify in the `r` parameter.

As an alternative to the `RectRgn` function, use the `SetRectRgn` function, which accepts as parameters four coordinates instead of a rectangle.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

The `RectRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HideMenuBar

QTCarbonShell

Declared In

`QuickdrawAPI.h`

ReserveEntry

Reserves or removes reservation from an entry in the current `GDevice` data structure's color table. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ReserveEntry (
   short index,
   Boolean reserve
);
```

Parameters

index

The index to the entry.

reserve

`True` to reserve the entry, `false` to remove the reservation.

Discussion

A reserved entry cannot be matched by another application's search function, and `Color2Index` (or other functions that depend on it such as `RGBForeColor`, `RGBBackColor`, and `SetCPixel`) never return that entry to another client. You could use this function to selectively protect a color for color table animation.

The `ReserveEntry` function copies the low byte of the `gdID` field of the current `GDevice` data structure into the low byte of the `ColorSpec.value` field of the color table when reserving an entry, and leaves the high byte alone. `ReserveEntry` acts like selective protection and does not allow any changes if the current `gdID` field is different than the one in the `ColorSpec.value` field of the reserved entry. If a requested match is already reserved, `ReserveEntry` returns a protection error. It can remove reservation from any entry, even if a requested match is already not reserved.

Carbon Porting Notes

This function does nothing useful on Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

RestoreEntries

Restores a selection of color table entries. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void RestoreEntries (
   CTabHandle srcTable,
   CTabHandle dstTable,
   ReqListRec *selection
);
```

Parameters

srcTable

The color table containing entries to be restored.

dstTable

The color table in which to restore the entries. If `dstTable` is `NULL`, or points to the current `GDevice` data structure's color table, `RestoreEntries` changes the device's color table and the hardware CLUT to these new colors.

selection

A pointer to the ReqListRec (page 2882) data structure. The entries to be restored are enumerated as offsets into a ColorTable data structure, not the contents of the ColorSpec.value field.

Discussion
The RestoreEntries function does not rebuild the inverse table.

If a request is beyond the end of the destination color table, RestoreEntries sets that position in the requestList data structure to colReqErr, and returns an error. RestoreEntries assumes that the color table specified by the srcTable parameter and the request list specified by the selection parameter have the same number of entries.

RestoreEntries does not change the color table's seed, so no invalidation occurs (which may cause RGBForeColor to act strangely). RestoreEntries ignores protection and reservation of color table entries.

You generally should use the Palette Manager to give your application its own set of colors; use of RestoreEntries should be limited to special-purpose applications. RestoreEntries allows you to change a color table without changing its ctSeed field. You can execute the application code and then use RestoreEntries to put the original colors back in. However, in some cases things in the background may appear in the wrong colors, since they were never redrawn. To avoid this, your application must build its own new inverse table and redraw the background. If you then use RestoreEntries, you should call the CTabChanged function to clean up correctly.

Carbon Porting Notes

This function does nothing useful on Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

RGBBackColor

Changes the background color. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void RGBBackColor (
   const RGBColor *color
);
```

Parameters

color

An RGBColor structure.

Discussion
If the current port is defined by a CGrafPort structure, QuickDraw supplies its rgbBkColor field with the RGB value that you specify in the color parameter, and places the pixel value most closely matching that color in the bkColor field. For indexed devices, the pixel value is an index to the current device's CLUT. F or direct devices, the value is the 16-bit or 32-bit equivalent to the RGB value.

If the current port is defined by a `GrafPort` structure, basic QuickDraw supplies its `fgColor` field with a color value determined by taking the high bit of each of the red, green, and blue components of the color that you supply in the `color` parameter. Basic QuickDraw uses that 3-bit number to select a color from its eight-color system.

You can also use Palette Manager functions to set the background color.

To determine the current background color, use the `GetBackColor` (page 2634) function.

Because a pixel pattern already contains color, QuickDraw ignores the background color and foreground colors when your application draws with a pixel pattern. Use the `PenPixPat` function to assign a pixel pattern to the foreground pattern used by the graphics pen. Use the `BackPixPat` function to assign a pixel pattern as the background pattern for the current color graphics port. Use the `FillCRect`, `FillCOval`, `FillCRoundRect`, `FillCArc`, `FillCRgn`, and `FillCPoly` functions to fill shapes with a pixel pattern.

Special Considerations

The `RGBBackColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

RGBForeColor

Changes the color of the "ink" used for framing and painting. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void RGBForeColor (
   const RGBColor *color
);
```

Parameters

color

An `RGBColor` structure.

Discussion

If the current port is defined by a `CGrafPort` structure, QuickDraw supplies its `rgbFgColor` field with the RGB value that you specify in the `color` parameter, and places the pixel value most closely matching that color in the `fgColor` field. For indexed devices, the pixel value is an index to the current device's CLUT. For direct devices, the value is the 16-bit or 32-bit equivalent to the RGB value.

If the current port is defined by a `GrafPort` structure, basic QuickDraw supplies its `fgColor` field with a color value determined by taking the high bit of each of the red, green, and blue components of the color that you supply in the `color` parameter. Basic QuickDraw uses that 3-bit number to select a color from its eight-color system.

You can also use Palette Manager functions to set the foreground color.

To determine the current foreground color, use the `GetForeColor` (page 2639) function.

QuickDraw ignores the foreground and background colors when your application draws with a pixel pattern. Assign a pixel pattern to the foreground pattern used by the graphics pen; by using the `BackPixPat` function to assign a pixel pattern as the background pattern for the current color graphics port; and by using the `FillCRect`, `FillCOval`, `FillCRoundRect`, `FillCArc`, `FillCRgn`, and `FillCPoly` functions to fill shapes with a pixel pattern.

Special Considerations

The `RGBForeColor` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`QuickdrawAPI.h`

RgnToHandle

```
void RgnToHandle (
   RgnHandle region,
   Handle flattenedRgnDataHdl
);
```

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SaveEntries

Saves a selection of color table entries. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SaveEntries (
   CTabHandle srcTable,
   CTabHandle resultTable,
   ReqListRec *selection
);
```

Parameters

srcTable
The color table containing entries to be saved. If you supply `NULL`, `SaveEntries` uses the current device's color table as the source.

resultTable

The color table in which to save the entries.

selection

A pointer to the ReqListRec (page 2882) data structure. The entries to be set are enumerated as offsets into a ColorTable data structure, not the contents of the ColorSpec.value field.) If an entry is not present in srcTable, then SaveEntries sets that position of the selection parameter to colReqErr, and that position of resultTable contains random values.

Discussion

If SaveEntries can not find one or more entries, then it posts an error code to QDError; however, for every entry in selection which is not colReqErr, the values in resultTable are valid. SaveEntries assumes that the color table specified by the srcTable parameter and the request list specified by the selection parameter have the same number of entries.

The output of SaveEntries is the same as the input for RestoreEntries, except for the order.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

QuickdrawAPI.h

ScalePt

Scales a height and width according to the proportions of two rectangles.

```
void ScalePt (
   Point *pt,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

pt

On input, a pointer to an initial height and width (specified in the two fields of a Point structure) to scale; on return, vertical and horizontal scaling factors derived by multiplying the height and width by ratios of the height and width of the rectangle in the srcRect parameter to the height and width of the rectangle in the dstRect parameter.

You do not pass coordinates in this parameter. Instead, you pass an initial height to scale in the v (or vertical) field of the Point structure, and you pass an initial width to scale in the h (or horizontal) field.

The ScalePt function scales these measurements by multiplying the initial height by the ratio of the height of the rectangle you specify in the dstRect parameter to the height of the rectangle you specify in the srcRect parameter, and by multiplying the initial width by the ratio of the width of the dstRect rectangle to the width of the srcRect rectangle.

srcRect

A rectangle. The ratio of this rectangle's height to the height of the rectangle in the dstRect parameter provides the vertical scaling factor, and the ratio of this rectangle's width to the width of the rectangle in the dstRect parameter provides the horizontal scaling factor.

dstRect

A rectangle compared to the rectangle in the `srcRect` parameter to determine vertical and horizontal scaling factors.

Discussion

The `ScalePt` function produces horizontal and vertical scaling factors from the proportions of two rectangles. Use `ScalePt`, for example, to scale the dimensions of the graphics pen.

Where the width of the `dstRect` rectangle is twice the width of the `srcRect` rectangle, and its height is three times the height of `srcRect`, `ScalePt` scales the width of the graphics pen from 3 to 6 and scales its height from 2 to 6.

Special Considerations

The minimum value `ScalePt` returns is (1,1).

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawAPI.h`

ScreenRes

Determines the resolution of the main device. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ScreenRes (
   short *scrnHRes,
   short *scrnVRes
);
```

Parameters

scrnHRes

On return, the number of horizontal pixels per inch displayed by the current device.

scrnVRes

On return, the number of vertical pixels per inch displayed by the current device.

Discussion

To determine the resolutions of all available graphics devices, examine their `GDevice` (page 2859) structures. The horizontal and vertical resolutions for a graphics device are stored in the `hRes` and `vRes` fields, respectively, of the `PixMap` structure for the device's `GDevice` structure.

Currently, QuickDraw and the Printing Manager always assume a screen resolution of 72 dpi.

Do not use the actual screen resolution as a scaling factor when drawing into a printing graphics port. Instead, always use 72 dpi as the scaling factor. See the Printing Manager documentation for more information about drawing into a printing graphics port.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ScrollRect

Scroll the pixels of a specified portion of a basic graphics port's bitmap (or a color graphics port's pixel map). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ScrollRect (
   const Rect *r,
   short dh,
   short dv,
   RgnHandle updateRgn
);
```

Parameters

r
The pointer to the rectangle defining the area to be scrolled.

dh
The horizontal distance to be scrolled.

dv
The vertical distance to be scrolled.

updateRgn
A handle to the region of the window that needs to be updated.

Discussion
The `ScrollRect` function shifts pixels that are inside the specified rectangle of the current graphics port. No other pixels or the bits they represent are affected. The pixels are shifted a distance of `dh` horizontally and `dv` vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified rectangle are not displayed, and the bits they represent are not saved. It is up to your application to save this data.

The empty area created by the scrolling is filled with the graphics port's background pattern, and the update region is changed to this filled area.

The `ScrollRect` function doesn't change the local coordinate system of the graphics port it simply moves the rectangle specified in the `r` parameter to different coordinates. Notice that `ScrollRect` doesn't move the graphics pen or the clipping region. However, because the document has moved, they're in different positions relative to the document.

By creating an update region for the window, `ScrollRect` forces an update event. After using `ScrollRect`, your application should use its own window-updating code to draw into the update region of the window.

The `ScrollRect` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SectRect

Determines whether two rectangles intersect.

```
Boolean SectRect (
   const Rect *src1,
   const Rect *src2,
   Rect *dstRect
);
```

Parameters

src1
The first of two rectangles to test for intersection.

src2
The second of two rectangles to test for intersection.

dstRect
On return, a pointer to the rectangle marking the intersection of the first two rectangles.

Return Value
`TRUE` if the specified rectangles intersect or `FALSE` if they do not.

Discussion
The `SectRect` function calculates the rectangle that delineates the intersection of the two rectangles you specify in the `src1` and `src2` parameters. Rectangles that touch at a line or a point are not considered intersecting, because their intersection rectangle (actually, in this case, an intersection line or point) does not enclose any pixels in the bit image.

If the rectangles do not intersect, the destination rectangle is set to (0,0,0,0). The `SectRect` function works correctly even if one of the source rectangles is also the destination.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SoftVDigX

Declared In
`QuickdrawAPI.h`

SectRegionWithPortClipRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SectRegionWithPortClipRegion (
   CGrafPtr port,
   RgnHandle ioRegion
);
```

Availability
Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SectRegionWithPortVisibleRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SectRegionWithPortVisibleRegion (
   CGrafPtr port,
   RgnHandle ioRegion
);
```

Availability
Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SectRgn

Calculates the intersection of two regions.

```
void SectRgn (
   RgnHandle srcRgnA,
   RgnHandle srcRgnB,
   RgnHandle dstRgn
);
```

Parameters
srcRgnA

A handle to the first of two regions whose intersection is to be determined.

srcRgnB

A handle to the second of two regions whose intersection is to be determined.

dstRgn

On return, a handle to the region holding the intersection area. If the regions do not intersect, or one of the regions is empty, `SectRgn` sets the destination to the empty region defined by the rectangle (0,0,0,0).

The `SectRgn` function does not create a destination region; you must have already allocated memory for it by using the `NewRgn` (page 2726) function.

The destination region may be one of the source regions, if desired.

Discussion

The SectRgn procedure calculates the intersection of the two regions whose handles you pass in the srcRgnA and srcRgnB parameters, and it places the intersection in the region whose handle you pass in the dstRgn parameter. If the regions do not intersect, or one of the regions is empty, SectRgn sets the destination to the empty region defined by the rectangle (0,0,0,0).

The SectRgn procedure does not create a destination region; you must have already allocated memory for it by using the NewRgn function.

The destination region may be one of the source regions, if desired.

Special Considerations

The `SectRgn` function may temporarily use heap space that's twice the size of the two input regions.

The `SectRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SeedCFill

Determines how far filling will extend to pixels matching the color of a particular pixel. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SeedCFill (
   const BitMap *srcBits,
   const BitMap *dstBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short seedH,
   short seedV,
   ColorSearchUPP matchProc,
   long matchData
);
```

Parameters

srcBits

The source image. If the image is in a pixel map, you must coerce its `PixMap` structure to a `BitMap` structure.

dstBits
On return, the destination mask.

srcRect
The rectangle of the source image.

dstRect
The rectangle of the destination image.

seedH
The horizontal position of the seed point.

seedV
The vertical position of the seed point.

matchProc
An optional color search function.

matchData
Data for the optional color search function.

Discussion
The `SeedCFill` function generates a mask showing where the pixels in an image can be filled from a starting point, like the paint pouring from the MacPaint paint-bucket tool. This mask is a bitmap filled with 1's to indicate all pixels adjacent to a seed point whose colors do not exactly match the `RGBColor` structure for the pixel at the seed point. You can then use this mask with the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions.

You specify a source image in the `srcBits` parameter and, in the `srcRect` parameter, specify a rectangle within that source image. You specify where to begin seeding in the `seedH` and `seedV` parameters, which must be the horizontal and vertical coordinates of a point in the local coordinate system of the source bitmap. By default, the 1's returned in the mask indicate all pixels adjacent to the seed point whose pixel values do not exactly match the pixel value of the pixel at the seed point. To use this default, set the `matchProc` and `matchData` parameters to 0.

In generating the mask, `SeedCFill` uses the `CopyBits` function to convert the source image to a 1-bit mask. The `SeedCFill` function installs a default color search function that returns 0 if the pixel value matches that of the seed point all other pixel values return 1's.

The `SeedCFill` function does not scale so the source and destination rectangles must be the same size. Calls to `SeedCFill` are not clipped to the current port and are not stored into QuickDraw pictures.

To customize `SeedCFill`, write your own color search function and point to it in the `matchProc` parameter; `SeedCFill` will then use your function instead of the default.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SeedFill

Determines how far filling will extend from a seeding point. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SeedFill (
   const void *srcPtr,
   void *dstPtr,
   short srcRow,
   short dstRow,
   short height,
   short words,
   short seedH,
   short seedV
);
```

Parameters

srcPtr

A pointer to the source bit image.

dstPtr

On input, a pointer to the destination bit image; upon return, a pointer to the bitmap containing the resulting mask.

srcRow

Row width of the source bitmap.

dstRow

Row width of the destination bitmap.

height

Height (in pixels) of the fill rectangle.

words

Width (in words) of the fill rectangle.

seedH

The horizontal offset (in pixels) at which to begin filling the destination bit image.

seedV

The vertical offset (in pixels) at which to begin filling the destination bit image.

Discussion

The `SeedFill` function produces a mask showing where bits in an image can be filled from a starting point, like the paint pouring from the MacPaint paint-bucket tool. The `SeedFill` returns this mask in the `dstPtr` parameter. This mask is a bitmap filled with 1's only where the pixels in the source image can be filled. You can then use this mask with the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions.

Point to the bit image you want to fill with the `srcPtr` parameter, which can point to the image's base address or a word boundary within the image. Specify a pixel height and word width with the `height` and `words` parameters to define a fill rectangle that delimits the area you want to fill. The fill rectangle can be the entire bit image or a subset of it. Point to a destination image with the `dstPtr` parameter. Specify the row widths of the source and destination bitmaps (their `rowBytes` values) with the `srcRow` and `dstRow` parameters. (The bitmaps can be different sizes, but they must be large enough to contain the fill rectangle at the origins specified by the `srcPtr` and `dstPtr` parameters.)

You specify where to begin filling with the `seedH` and `seedV` parameters: they specify a horizontal and vertical offset in pixels from the origin of the image pointed to by the `srcPtr` parameter. The `SeedFill` function calculates contiguous pixels from that point out to the boundaries of the fill rectangle, and it stores the result in the bit image pointed to by the `dstPtr` parameter.

Calls to `SeedFill` are not clipped to the current port and are not stored into QuickDraw pictures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetCCursor

Specifies a color cursor for display on the screen. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetCCursor (
   CCrsrHandle cCrsr
);
```

Parameters

cCrsr
A handle to the color cursor to be displayed.

Discussion
At the time the cursor is set, it is expanded to the current screen depth so that it can be drawn rapidly. You must call `GetCCursor` before you call `SetCCursor`; however, you can make several subsequent calls to `SetCCursor` once `GetCCursor` creates the `CCrsr` structure.

If your application has changed the cursor's data or its color table, it must also invalidate the `crsrXValid` and `crsrID` fields of the `CCrsr` structure before calling `SetCCursor`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetClientID

Sets the `gdID` field in the current `GDevice` data structure to identify this client program to its search and complement functions. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetClientID (
   short id
);
```

Parameters

id
The ID to be set in the device data structure.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetClip

Changes the clipping region of the current graphics port (basic or color) to a region you specify. (**Deprecated.** Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetClip (
   RgnHandle rgn
);
```

Parameters

rgn

A handle to a region. The `SetClip` function makes this region the clipping region of the current graphics port. The `SetClip` function doesn't change the region handle, but instead affects the clipping region itself.

Discussion
Since `SetClip` copies the specified region into the current graphics port's clipping region, any subsequent changes you make to the region specified in the `rgn` parameter do not affect the clipping region of the graphics port.

The initial clipping region of a graphics port is an arbitrarily large rectangle. You can set the clipping region to any arbitrary region, to aid you in drawing inside the graphics port—for example, to avoid drawing over scroll bars when drawing into a window, you could define a clipping region that excludes the scroll bars.

You can use the `GetClip` and `SetClip` functions to preserve the current clipping region: use `GetClip` to save the current port's clipping region, and use `SetClip` to restore it.

All other system software functions preserve the current clipping region.

The `SetClip` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.k.h`

SetCPixel

Sets the color of an individual pixel to the color that most closely matches the RGB color that you specify in the `cPix` parameter. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetCPixel (
   short h,
   short v,
   const RGBColor *cPix
);
```

Parameters

h

The horizontal coordinate of the point at the upper-left corner of the pixel.

v

The vertical coordinate of the point at the upper-left corner of the pixel.

cPix

An `RGBColor` structure.

Discussion

On an indexed color system, the `SetCPixel` function sets the pixel value to the index of the best-matching color in the current device's CLUT. In a direct environment, the `SetCPixel` function sets the pixel value to a 16-bit or 32-bit direct pixel value.

To determine the color of an individual pixel, use the `GetCPixel` (page 2636) function.

Special Considerations

The `SetCPixel` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetCursor

Sets the current cursor. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetCursor (
    const Cursor * crsr
);
```

Parameters

crsr

A `Cursor` (page 2854) structure for the cursor to be displayed.

Discussion

If the cursor is hidden, it remains hidden and attains its new appearance only when it's uncovered. If the cursor is already visible, it changes to the new appearance immediately.

You need to use the `InitCursor` (page 2671) function to initialize the standard arrow cursor and make it visible on the screen before you call `SetCursor` to change the cursor's appearance.

To display a color cursor, use the `SetCCursor` (page 2790) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetCursorComponent

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr SetCursorComponent (
   ComponentInstance ci
);
```

Return Value

Carbon Porting Notes

This function is not implemented on Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetDeviceAttribute

Sets the attribute bits of a `GDevice` structure. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetDeviceAttribute (
   GDHandle gdh,
   short attribute,
   Boolean value
);
```

Parameters

gdh
A handle to a `GDevice` structure.

attribute
One of the specific constants, which represent bits in the `gdFlags` field of a `GDevice` structure. See GDevice (page 2859) for the values you can use in this parameter.

value
A value of either 0 or 1 for the flag bit specified in the `attribute` parameter.

Discussion
For the graphics device specified in the `gdh` parameter, the `SetDeviceAttribute` function sets the flag bit specified in the `attribute` parameter to the value specified in the `value` parameter.

Your application should never directly change the `gdFlags` field of the `GDevice` structure; instead, use only the `SetDeviceAttribute` function.

Special Considerations

The `SetDeviceAttribute` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetEmptyRgn

Sets an existing region to be empty.

```
void SetEmptyRgn (
   RgnHandle rgn
);
```

Parameters

rgn
A handle to the region to be made empty.

Discussion
The `SetEmptyRgn` function destroys the previous structure of the region whose handle you pass in the `rgn` parameter; it then sets the new structure to the empty region defined by the rectangle (0,0,0,0).

Special Considerations

The `SetEmptyRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetEntries

Sets a group of color table entries for the current `GDevice` data structure. This function is used by system software and your application should not need to call it. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetEntries (
   short start,
   short count,
   CSpecArray aTable
);
```

Parameters

start

The index of the first entry to be changed.

count

The number of entries to be changed. Note that all values are zero-based; for example, to set three entries, pass 2 in the count parameter.

aTable

An array of `ColorSpec` data structures containing the colors to be used. Directly specify a `cSpecArray` structure, not the beginning of a color table. The `ColorSpec.value` fields of the entries must be in the logical range for the target device's assigned pixel depth. Thus, with a 4-bit pixel size, the `ColorSpec.value` fields should be in the range 1 to 15. With an 8-bit pixel size, the range is 0 to 255.

Discussion

Instead of using `SetEntries`, you should use the Palette Manager function `SetEntryColor` to allow your application to run in a multiscreen or multitasking environment.

The `SetEntries` positional information works in logical space rather than in the actual memory space used by the hardware. Requesting a change at the fourth position in the color table may not modify the fourth color table entry in the hardware, but it does correctly change the color on the screen for any pixels with a value of 4 in the video card. The `SetEntries` mode characterized by a start position and a length is called sequence mode. In this case, `SetEntries` sequentially loads new colors into the hardware in the same order as they appear in the `aTable` parameter, copies the `clientID` fields for changed color table entries from the current `GDevice` data structure's `gdID` field, and ignores the `ColorSpec.value` fields.

The other `SetEntries` mode is called index mode. It allows the `cSpecArray` structure to specify where the data will be installed on an entry-by-entry basis. To use this mode, pass –1 for the start position, with a valid count and a pointer to the `cSpecArray` data structure. Each entry is installed into the color table at the position specified by the `ColorSpec.value` field of each entry in the `cSpecArray` data structure. In the current `GDevice` data structure's color table, the `ColorSpec.value` fields of all changed entries are assigned the `GDevice` data structure's `gdID` value.

When the Color Manager changes color table entries, it invalidates all cached fonts, and changes the color table's seed number so that the next drawing operation triggers the Color Manager to rebuild the inverse table. If any of the requested entries are protected or out of range, the Color Manager returns a protection error, and nothing happens. The Color Manager changes a reserved entry only if the current `gdID` field of the current `GDevice` data structure matches the low byte of the intended `ColorSpec.value` field in the color table.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetGDevice

Sets a `GDevice` structure as the current device. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetGDevice (
   GDHandle gd
);
```

Parameters

gd

A handle to a `GDevice` structure.

Discussion

Your application won't generally need to use this function, because when your application draws into a window on one or more screens, Color QuickDraw automatically switches `GDevice` structures as appropriate; and when your application needs to draw into an offscreen graphics world, it can use the `SetGWorld` function to set the graphics port as well as the `GDevice` structure for the offscreen environment. However, if your application uses the `SetPort` function instead of the `SetGWorld` function to set the graphics port to or from an offscreen graphics world, then your application must use `SetGDevice` in conjunction with `SetPort`.

A handle to the currently active device is kept in the global variable `TheGDevice`.

Special Considerations

The `SetGDevice` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetGWorld

Changes the current graphics port (basic, color, or offscreen). (**Deprecated.** Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetGWorld (
   CGrafPtr port,
   GDHandle gdh
);
```

Parameters

port

A pointer to an offscreen graphics world, color graphics port, or basic graphics port. Specify values of type `GrafPtr`, `CGrafPtr`, or `GWorldPtr`, depending on whether you want to set the current graphics port to be a basic graphics port, color graphics port, or offscreen graphics world. Any drawing your application performs then occurs in this graphics port.

gdh

A handle to a `GDevice` structure. If you pass a pointer to an offscreen graphics world in the `port` parameter, set this parameter to `NULL`, because `SetGWorld` ignores this parameter and sets the current device to the device attached to the offscreen graphics world.

Discussion
The `SetGWorld` function sets the current graphics port to the one specified by the `port` parameter and—unless you set the current graphics port to be an offscreen graphics world—sets the current device to that specified by the `gdh` parameter.

Special Considerations

The `SetGWorld` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ASCIIMoviePlayerSample
QTCarbonShell
Simple DrawSprocket

Declared In
`ImageCompression.k.h`

SetOrigin

Changes the coordinates of the window origin of the port rectangle of the current graphics port (basic or color). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetOrigin (
   short h,
   short v
);
```

Parameters

h

The horizontal coordinate of the upper-left corner of the port rectangle.

v

The vertical coordinate of the upper-left corner of the port rectangle.

Discussion
The `SetOrigin` function changes the coordinates of the upper-left corner of the current graphics port's port rectangle to the values supplied by the `h` and `v` parameters. All other points in the current graphics port's local coordinate system are calculated from this point. All subsequent drawing and calculation functions use the new coordinate system.

The `SetOrigin` function does not affect the screen; it does, however, affect where subsequent drawing inside the graphics port appears. The `SetOrigin` function does not offset the coordinates of the clipping region or the graphics pen, which therefore change position on the screen (unlike the boundary rectangle, port rectangle, and visible region, which don't change position onscreen).

Because `SetOrigin` does not move the window's clipping region, use the `GetClip` function to store your clipping region immediately after your first call to `SetOrigin`—if you use clipping regions in your windows. Before calling your own window-drawing function, use the `ClipRect` function to define a new clipping region—to avoid drawing over your scroll bars, for example. After calling your own window-drawing function, use the `SetClip` function to restore the original clipping region. You can then call `SetOrigin` again to restore the window origin to a horizontal coordinate of 0 and a vertical coordinate of 0 with your original clipping region intact.

All other functions in the Macintosh Toolbox and Operating System preserve the local coordinate system of the current graphics port. The `SetOrigin` function is useful for readjusting the coordinate system after a scrolling operation.

Note that the Window Manager and Control Manager always assume the window's upper-left point has a horizontal coordinate of 0 and a vertical coordinate of 0 when they draw in a window. Therefore, if you use `SetOrigin` to change the window origin, be sure to use `SetOrigin` again to return the window origin to a horizontal coordinate of 0 and a vertical coordinate of 0 before using any Window Manager or Control Manager functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPenState

Restores the state of the graphics pen that was saved with the `GetPenState` function. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPenState (
   const PenState *pnState
);
```

Parameters

`pnState`

A `PenState` structure previously created with the `GetPenState` function. The `SetPenState` function sets the graphics pen's location, size, pattern, and pattern mode in the current graphics port to the values stored in this structure.

Discussion
This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPixelsState

Restores an offscreen pixel image to the state that you saved with the `GetPixelsState` function. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPixelsState (
   PixMapHandle pm,
   GWorldFlags state
);
```

Parameters

pm

A handle to an offscreen pixel map.

state

Flags, which you usually save with the `GetPixelsState` function. You can use either of the constants `pixelsPurgeable` or `pixelsLocked here`.

Because only an unlocked memory block can be purged, `SetPixelsState` calls the `UnlockPixels` (page 2829) and `AllowPurgePixels` (page 2573) functions if the `state` parameter specifies the `pixelsPurgeable` flag. If the `state` parameter does not specify the `pixelsPurgeable` flag, `SetPixelsState` makes the base address for the offscreen pixel image unpurgeable.

If the `state` parameter does not specify the `pixelsLocked` flag, `SetPixelsState` allows the base address for the offscreen pixel image to be moved.

Discussion

The `SetPixelsState` function changes the state of the memory allocated for an offscreen pixel image to the state indicated in the `state` parameter.

After using `GetPixelsState` and before using `SetPixelsState`, your application can temporarily alter the offscreen graphics world by using the `AllowPurgePixels` (page 2573) function to temporarily mark the memory block for its offscreen pixel map as purgeable, the `NoPurgePixels` (page 2729) function to make it unpurgeable, the `LockPixels` (page 2704) function to prevent it from being moved, and the `UnlockPixels` (page 2829) function to unlock it.

Special Considerations

The `SetPixelsState` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QDOffscreen.h`

SetPort

Changes the current graphics port (basic or color). (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
virtual void SetPort (
   void *port
);
```

Parameters

port

A pointer to a `GrafPort` structure. Typically, you pass a pointer to a `GrafPort` structure that you previously saved with the `GetPort` function. The `SetPort` function sets this structure to be the current graphics port.

Discussion

All QuickDraw drawing functions affect the bitmap of, and use the local coordinate system of, the current graphics port. Each graphics port has its own graphics pen and text characteristics, which remain unchanged when the graphics port isn't selected as the current graphics port.

When your application runs in Color QuickDraw or uses offscreen graphics worlds, it should use the `SetGWorld` function instead of `SetPort`. The `SetGWorld` function restores the current graphics port for basic and color graphics ports as well as offscreen graphics worlds.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

HideMenuBar

QTCarbonShell

Declared In

`QuickdrawAPI.h`

SetPortBackPixPat

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortBackPixPat (
   CGrafPtr port,
   PixPatHandle backPattern
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetPortBits

Sets the bitmap for the current basic graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortBits (
   const BitMap *bm
);
```

Parameters

bm

A pointer to the `BitMap` structure to set for the current graphics port. Be sure to prepare all fields of the `BitMap` structure before you call `SetPortBits`.

Discussion

You should never need to use this function. This function, created for early versions of QuickDraw, allows you to perform all normal drawing and calculations on a buffer other than the screen—for example, copying a small offscreen image onto the screen with the `CopyBits` function. However, instead of using `SetPortBits`, you should use the more powerful offscreen graphics capabilities.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetPortBounds

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortBounds (
   CGrafPtr port,
   const Rect *rect
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetPortClipRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortClipRegion (
   CGrafPtr port,
   RgnHandle clipRgn
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortCustomXFerProc

(Deprecated in Mac OS X v10.4.)

Not recommended

```
OSErr SetPortCustomXFerProc (
   CGrafPtr port,
   CustomXFerProcPtr proc,
   UInt32 flags,
   UInt32 refCon
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortFillPixPat

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortFillPixPat (
   CGrafPtr port,
   PixPatHandle penPattern
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

SetPortFracHPenLocation

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortFracHPenLocation (
   CGrafPtr port,
   short pnLocHFrac
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

SetPortGrafProcs

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortGrafProcs (
   CGrafPtr port,
   CQDProcsPtr procs
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

SetPortOpColor

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortOpColor (
   CGrafPtr port,
   const RGBColor *opColor
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetPortPenMode

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortPenMode (
   CGrafPtr port,
   SInt32 penMode
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SetPortPenPixPat

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortPenPixPat (
   CGrafPtr port,
   PixPatHandle penPattern
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortPenSize

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortPenSize (
   CGrafPtr port,
   Point penSize
);
```

Carbon Porting Notes

Use this new accessor function in place of direct access to structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortPix

Sets the pixel map for the current color graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortPix (
   PixMapHandle pm
);
```

Parameters

pm

A handle to the `PixMap` structure.

Discussion
The `SetPortPix` function replaces the `portPixMap` field of the current `CGrafPort` structure with the handle you specify in the `pm` parameter.

Typically, your application does not need to call this function.

The `SetPortPix` function is analogous to the basic QuickDraw function `SetPortBits`, which sets the bitmap for the current basic graphics port. The `SetPortPix` function has no effect when used with a basic graphics port. Similarly, `SetPortBits` has no effect when used with a color graphics port.

Both `SetPortPix` and `SetPortBits` allow you to perform drawing and calculations on a buffer other than the screen. However, instead of using these functions, use the offscreen graphics capabilities.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortTextFace

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortTextFace (
   CGrafPtr port,
   StyleParameter face
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortTextFont

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortTextFont (
   CGrafPtr port,
   short txFont
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortTextMode

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortTextMode (
   CGrafPtr port,
   short mode
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortTextSize

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortTextSize (
   CGrafPtr port,
   short txSize
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPortVisibleRegion

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetPortVisibleRegion (
   CGrafPtr port,
   RgnHandle visRgn
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetPt

Assigns two coordinates to a point.

```
void SetPt (
   Point *pt,
   short h,
   short v
);
```

Parameters

pt

A pointer to the point to be given new coordinates. On return, this point is assigned the horizontal coordinate you specify in the `h` parameter and the vertical coordinate you specify in the `v` parameter.

h

The horizontal value of the new coordinates.

v

The vertical value of the new coordinates.

Discussion
The SetPt procedure assigns the horizontal coordinate specified in the h parameter and the vertical coordinate specified in the v parameter to the point returned in the pt parameter.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawAPI.h`

SetQDError

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetQDError (
   OSErr err
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetQDGlobalsArrow

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetQDGlobalsArrow (
   const Cursor *arrow
);
```

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetQDGlobalsRandomSeed

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetQDGlobalsRandomSeed (
   long randomSeed
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetRect

Assigns coordinates to a rectangle.

```
void SetRect (
    Rect * r,
    short left,
    short top,
    short right,
    short bottom
);
```

Parameters

r
A pointer to the rectangle to set.

left
The horizontal coordinate of the new upper-left corner of the rectangle.

top
The vertical coordinate of the new upper-left corner of the rectangle.

right
The horizontal coordinate of the new lower-right corner of the rectangle.

bottom
The vertical coordinate of the new lower-right corner of the rectangle.

Discussion
The `SetRect` function assigns the coordinates you specify in the `left`, `top`, `right`, and `bottom` parameters to the rectangle that you specify in the `r` parameter. This function is provided to help you shorten your program text. If you want a more readable text, at the expense of source text length, you can instead assign integers (or points) directly into the fields of a `Rect` structure.

You can use a rectangle to specify locations and sizes for various graphics operations.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
HID Calibrator
WhackedTV

Declared In
`QuickdrawAPI.h`

SetRectRgn

Changes the structure of an existing region to that of a rectangle.

```
void SetRectRgn (
    RgnHandle rgn,
    short left,
    short top,
    short right,
    short bottom
);
```

Parameters

rgn
A handle to the region to restructure as a rectangle.

left
The horizontal coordinate of the upper-left corner of the rectangle to set as the new region.

top
The vertical coordinate of the upper-left corner of the rectangle to set as the new region.

right
The horizontal coordinate of the lower-right corner of the rectangle to set as the new region.

bottom
The vertical coordinate of the lower-right corner of the rectangle to set as the new region.

Discussion
The `SetRectRgn` function destroys the previous structure of the region whose handle you pass in the `rgn` parameter, and it then sets the new structure to the rectangle that you specify in the `left`, `top`, `right`, and `bottom` parameters. If you specify an empty rectangle (that is, `right` is greater than or equal to `left` or `bottom = top`), the `SetRectRgn` function sets the region to the empty region defined by the rectangle (0,0,0,0).

As an alternative to the `SetRectRgn` function, you can change the structure of an existing region to that of a rectangle by using the `RectRgn` (page 2776) function, which accepts as a parameter a rectangle instead of four coordinates.

If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Special Considerations

The `SetRectRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetStdCProcs

Obtains a `CQDProcs` structure with fields that point to QuickDraw's standard low-level functions, which you can modify to change QuickDraw's standard low-level behavior. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetStdCProcs (
   CQDProcs *procs
);
```

Parameters

procs

Upon completion, a `CQDProcs` structure with fields that point to QuickDraw's standard low-level functions. You can change one or more fields to point to your own functions and then set the color graphics port to use this modified `CQDProcs` (page 2852) structure.

Discussion
For each shape that QuickDraw can draw, certain functions perform basic graphics operations on the shape: framing, painting, erasing, inverting, and filling. These functions, in turn, call a low-level drawing function for the shape.

The `grafProcs` field determines which low-level functions are called. If that field contains a value of `NULL`, the standard functions are called. You can set the `grafProcs` field to point to a structure of pointers to your own functions, and either completely override the standard ones or call them after your functions have modified their parameters as necessary.

The `SetStdCProcs` function sets all the fields of the `CQDProcs` structure to point to the standard functions. You can then reset the ones with which you are concerned.

The functions you install in the `CDQProcs` structure must have the same calling sequences as the standard basic QuickDraw functions.

When drawing in a color graphics port, your application must always use `SetStdCProcs` instead of `SetStdProcs`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SetStdProcs

Obtains a `QDProcs` structure with fields that point to basic QuickDraw's standard low-level functions, which you can modify to point to your own functions. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void SetStdProcs (
   QDProcs *procs
);
```

Parameters

procs

On return, a pointer to a `QDProcs` structure with fields that point to basic QuickDraw's standard low-level functions. You can change one or more fields of this structure to point to your own functions and then set the basic graphics port to use this modified `QDProcs` structure. By changing these pointers, you can install your own functions, and either completely override the standard ones or call them after your functions have modified their parameters as necessary.

Discussion
The functions you install in this `QDProcs` structure must have the same calling sequences as the standard functions.

Special Considerations

The Color QuickDraw function `SetStdCProcs` is analogous to the `SetStdProcs` function, which you should use with computers that support only basic QuickDraw. When drawing in a color graphics port, your application must always use `SetStdCProcs` instead of `SetStdProcs`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

ShieldCursor

Hides the cursor in a rectangle.

```
void ShieldCursor (
   const Rect *shieldRect,
   Point offsetPt
);
```

Parameters

shieldRect

A rectangle in which the cursor is hidden whenever the cursor intersects the rectangle. The rectangle may be specified in global or local coordinates. If you are using global coordinates, pass (0,0) in the `offsetPt` parameter. If you are using the local coordinates of a graphics port, pass the coordinates for the upper-left corner of the graphics port's boundary rectangle in the `offsetPt` parameter.

offsetPt

A point value for the offset of the rectangle. Like the basic QuickDraw function `LocalToGlobal`, the `ShieldCursor` function offsets the coordinates of the rectangle by the coordinates of this point.

Discussion

If the cursor and the given rectangle intersect, `ShieldCursor` hides the cursor. If they do not intersect, the cursor remains visible while the mouse is not moving, but is hidden when the mouse moves. Use this function with a feature such as QuickTime to display content in a specified rectangle. When a QuickTime movie is animating, the cursor should not be visible in front of the movie.

The `ShieldCursor` function decrements the cursor level and should be balanced by a call to the `ShowCursor` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

ShowCursor

Displays a cursor hidden by the `HideCursor` or `ShieldCursor` functions.

```
void ShowCursor (
   void
);
```

Discussion

`ShowCursor` increments the cursor level, which has been decremented by the `HideCursor` (page 2668) or `ShieldCursor` (page 2813) function and displays the cursor on the screen when the level is 0. A call to the `ShowCursor` function balances each previous call to the `HideCursor` or `ShieldCursor` function. The level is not incremented beyond 0, so extra calls to `ShowCursor` have no effect.

Low-level interrupt-driven functions link the cursor with the mouse position, so that if the cursor level is 0 and visible, the cursor automatically follows the mouse.

If the cursor has been changed with the `SetCursor` (page 2792) function while hidden, `ShowCursor` displays the new cursor.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`QuickdrawAPI.h`

ShowPen

Changes the ink of a graphics pen from invisible to visible, making pen drawing appear on the screen. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void ShowPen (
   void
);
```

Discussion
`ShowPen` is called by the functions `CloseRgn` (page 2582) , `ClosePoly` (page 2581) , and `ClosePicture`.

The `ShowPen` function increments the `pnVis` field of the current graphics port. For 0 or positive values, the pen drawing shows on the screen.

For example, if you have used the `HidePen` function to decrement the `pnVis` field from 0 to –1, use the `ShowPen` function to make its value 0 so that QuickDraw resumes drawing on the screen. Subsequent calls to `ShowPen` increment `pnVis` beyond 0, so every call to `ShowPen` should be balanced by a call to `HidePen`.

This pen-manipulation function uses the local coordinate system of the current graphics port. Remember that each graphics port has its own pen, the state of which is stored in several fields of its `GrafPort` or `CGrafPort` structure. If you draw in one graphics port, change to another, and return to the first, the pen for the first graphics port has the same state as when you left it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SlopeFromAngle

Converts an angle value to a slope value.

```
Fixed SlopeFromAngle (
   short angle
);
```

Parameters

angle

The angle, expressed in clockwise degrees from 12 o'clock and treated MOD 180. (90 degrees is thus at 3 o'clock and –90 degrees is at 9 o'clock.

Return Value

The slope corresponding to the angle specified in the angle parameter. Slopes are defined as Dx/Dy, the horizontal change divided by the vertical change between any two points on a line with the given angle. The negative y-axis is defined as being at 12 o'clock, and the positive y-axis at 6 o'clock. The x-axis is defined as usual, with the positive side defined as being at 3 o'clock.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

QuickdrawAPI.h

StdArc

QuickDraw's standard low-level function for drawing an arc or a wedge. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdArc (
   GrafVerb verb,
   const Rect *r,
   short startAngle,
   short arcAngle
);
```

Parameters

verb

The action to perform. See "Verb Constants" (page 2904).

r

The rectangle to contain the arc.

startAngle

The beginning angle.

arcAngle

The ending angle.

Discussion

Using the action specified in the verb parameter, the StdArc function draws an arc or wedge of the oval that fits inside the rectangle specified in the r parameter. The arc or wedge is bounded by the radii specified in the startAngle and arcAngle parameters.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdArc` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

StdBits

QuickDraw's standard low-level function for transferring bits and pixels. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdBits (
   const BitMap *srcBits,
   const Rect *srcRect,
   const Rect *dstRect,
   short mode,
   RgnHandle maskRgn
);
```

Parameters

srcBits

A pointer to a bitmap or pixel map containing the image to copy.

srcRect

A pointer to the source rectangle.

dstRect

The destination rectangle.

mode

The source mode for the copy.

maskRgn

A handle to a region acting as a mask for the transfer.

Discussion

The `StdBits` function transfers a bit or pixel image between the bitmap or pixel map specified in the `srcBits` parameter and bitmap of the current graphics port, just as if the `CopyBits` function were called with the same parameters and with a destination bitmap equal to (* `thePort).portBits`.

You should only call this low-level function from your customized QuickDraw functions.

See `CopyBits` (page 2584) for a discussion of the destination bitmap and of the `srcBits`, `srcRect`, `dstRect`, `mode`, and `maskRgn` parameters

Special Considerations

The `StdBits` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

StdComment

QuickDraw's standard low-level function for processing a picture comment. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdComment (
   short kind,
   short dataSize,
   Handle dataHandle
);
```

Parameters

kind
The type of comment.

dataSize
The size of additional data, in bytes.

dataHandle
A handle to additional data.

Discussion
If there's no additional data for the comment, the value of the `dataHandle` parameter is `NULL` and the value of the `dataSize` parameter is 0. The `StdComment` function simply ignores the comment.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdComment` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

StdGetPic

QuickDraw's standard low-level function for retrieving information from the definition of a picture. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdGetPic (
   void *dataPtr,
   short byteCount
);
```

Parameters

dataPtr

On return, a pointer to the collected picture data.

byteCount

The size of the picture data.

Discussion

The `StdGetPic` function retrieves from the definition of the currently open picture the next number of bytes as specified in the `byteCount` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

StdLine

QuickDraw's standard low-level function for drawing a line. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdLine (
   Point newPt
);
```

Parameters

newPt

The point to which to draw the line.

Discussion

The `StdLine` function draws a line from the current pen location to the location (in local coordinates) specified in the `newPt` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdLine` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

StdOpcode

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdOpcode (
   const Rect *fromRect,
   const Rect *toRect,
   UInt16 opcode,
   SInt16 version
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

StdOval

QuickDraw's standard low-level function for drawing an oval. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdOval (
   GrafVerb verb,
   const Rect *r
);
```

Parameters

verb
The action to perform. See "Verb Constants" (page 2904).

r
The rectangle to contain the oval.

Discussion
The `StdOval` function draws an oval inside the given rectangle specified in the `r` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdOval` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

StdPoly

QuickDraw's standard low-level function for drawing a polygon. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdPoly (
   GrafVerb verb,
   PolyHandle poly
);
```

Parameters

verb

The action to perform. See "Verb Constants" (page 2904).

poly

A handle to the polygon data.

Discussion
The `StdPoly` function draws the polygon specified in the `poly` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdPoly` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

StdPutPic

QuickDraw's standard low-level function for saving information as the definition of a picture. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdPutPic (
   const void *dataPtr,
   short byteCount
);
```

Parameters

dataPtr

A pointer to the collected picture data.

byteCount

The size of the picture data.

Discussion
The `StdPutPic` function saves as the definition of the currently open picture the drawing commands stored in the data structure pointed to by the `dataPtr` parameter, starting with the first byte and continuing for the next number of bytes as specified in the `byteCount` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdPutPic` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

StdRect

QuickDraw's standard low-level function for drawing a rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdRect (
   GrafVerb verb,
   const Rect *r
);
```

Parameters

verb
The action to perform. See "Verb Constants" (page 2904).

r
The rectangle to draw.

Discussion
The `StdRect` function draws the rectangle specified in the `r` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

StdRgn

QuickDraw's standard low-level function for drawing a region. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdRgn (
   GrafVerb verb,
   RgnHandle rgn
);
```

Parameters

verb

The action to perform. See "Verb Constants" (page 2904).

rgn

A handle to the region data.

Discussion

The `StdRgn` function draws the region specified in the `rgn` parameter according to the action specified in the `verb` parameter.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

StdRRect

QuickDraw's standard low-level function for drawing a rounded rectangle. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StdRRect (
   GrafVerb verb,
   const Rect *r,
   short ovalWidth,
   short ovalHeight
);
```

Parameters

verb

The action to perform. See "Verb Constants" (page 2904).

r

The rectangle to draw.

ovalWidth

The width diameter for the corner oval.

ovalHeight

The height diameter for the corner oval.

Discussion

The `StdRRect` function draws the rounded rectangle specified in the `r` parameter according to the action specified in the `verb` parameter. The `ovalWidth` and `ovalHeight` parameters specify the diameters of curvature for the corners.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdRRect` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

StuffHex

Sets byte values into memory. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void StuffHex (
   void *thingPtr,
   ConstStr255Param s
);
```

Parameters

thingPtr

A pointer to any data structure in memory. If `thingPtr` is an odd address, then `thingPtr` is interpreted as pointing to the next word boundary.

s

A string of characters representing hexadecimal digits. All characters in this string must be hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Otherwise, `StuffHex` may set bytes in the data structure pointed to by `thingPtr` to arbitrary values. If there are an odd number of characters in the string, the last character is ignored.

Discussion

The `StuffHex` function sets bytes in memory beginning with that byte specified by the parameter `thingPtr`. The total number of bytes set is equivalent to half the length of the string, ignoring the last character if the number of characters is odd.

Each byte to be set corresponds to two characters in the string. These characters should represent hexadecimal digits. For example, the string ‘`D41A`’ results in 2 bytes being set to the values $D4 and $1A, respectively.

To copy a range of bytes from one memory location to another, you should ordinarily use the Memory Manager function, `BlockMove`.

Special Considerations

The `StuffHex` function does no range checking to ensure that bytes being set are within the bounds of a certain data structure. If you do not use `StuffHex` carefully, you may change memory in the partition of your application or another application in unpredictable ways.

Although the `StuffHex` function sets the value of individual bytes, it does not move relocatable blocks. Thus, you can call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

SubPt

Subtracts the coordinates of one point from another.

```
void SubPt (
   Point src,
   Point *dst
);
```

Parameters

`src`

A point, the coordinates of which are to be subtracted from the coordinates of the point specified in the `dst` parameter.

`dst`

The address of a point. Upon completion, the coordinates of this point contain the differences between the coordinates of the two points specified in the entry parameters.

If you pass `NULL` in the `dst` parameter, this function sets the `QDError` result code to `paramErr` and returns.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawAPI.h`

SwapPortPicSaveHandle

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Handle SwapPortPicSaveHandle (
   CGrafPtr port,
   Handle inPicSaveHdl
);
```

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SwapPortPolySaveHandle

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Handle SwapPortPolySaveHandle (
   CGrafPtr port,
   Handle inPolySaveHdl
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SwapPortRegionSaveHandle

(Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Handle SwapPortRegionSaveHandle (
   CGrafPtr port,
   Handle inRegionSaveHdl
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

SyncCGContextOriginWithPort

Synchronizes the origin in a Quartz context with the lower-left corner of the associated graphics port. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSStatus SyncCGContextOriginWithPort (
   CGContextRef inContext,
   CGrafPtr port
);
```

Parameters

context

A Quartz context associated with a graphics port. You can obtain such a context by calling `QDBeginCGContext` (page 2756).

port

The graphics port associated with the context.

Return Value

A result code. If `noErr`, the context's origin was successfully changed.

Discussion

If you're using Quartz 2D to draw in a graphics port and `SetOrigin` (page 2797) is called to change the port's origin, you can call this function to maintain the correspondence between the context's origin and the lower-left corner of the `portBounds` rectangle.

When you call this function:

1. The current transformation matrix (CTM) is reset to its default values. Any changes you made to the CTM prior to calling this function are lost.
2. The CTM is translated to establish the new origin, taking the port's current origin into account.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

TestDeviceAttribute

Determines whether the flag bit for an attribute has been set in the `gdFlags` field of a `GDevice` structure. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
Boolean TestDeviceAttribute (
   GDHandle gdh,
   short attribute
);
```

Parameters

gdh

A handle to a `GDevice` structure.

attribute

One of the specific constants, which represent bits in the `gdFlags` field of a `GDevice` structure. See "Device Attribute Constants" (page 2887) for a description of the values you can use in this parameter.

Return Value

`TRUE` if the bit of the graphics device attribute specified in the `attribute` parameter is set to 1. Otherwise, `TestDeviceAttribute` returns `FALSE`.

Discussion

Use the `SetDeviceAttribute` (page 2793) function to change any of the flags tested by the `TestDeviceAttribute` function.

Special Considerations

The `TestDeviceAttribute` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawAPI.h`

UnionRect

Calculates the smallest rectangle that encloses two rectangles.

```
void UnionRect (
    const Rect * src1,
    const Rect * src2,
    Rect * dstRect
);
```

Parameters

src1

The first of two rectangles to enclose.

src2

The second of two rectangles to enclose.

dstRect

On return, a pointer to the smallest rectangle that encloses both of the rectangles you specify in the `src1` and `src2` parameters. One of the source rectangles may also be the destination.

Discussion
If the points or rectangles supplied to this function are defined in a graphics port other than your current graphics port, you must convert them to the local coordinate system of your current graphics port. You can accomplish this by using the `SetPort` function to change to the graphics port containing the points or rectangles, using the `LocalGlobal` function to convert their locations to global coordinates, using `SetPort` to return to your starting graphics port, and then using the `GlobalToLocal` function to convert the locations of points or rectangles to the local coordinates of your current graphics port.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawAPI.h`

UnionRgn

Calculates the union of two regions.

```
void UnionRgn (
    RgnHandle srcRgnA,
    RgnHandle srcRgnB,
    RgnHandle dstRgn
);
```

Parameters

srcRgnA
A handle to the first of two regions whose union is to be determined.

srcRgnB
A handle to the second of two regions whose union is to be determined.

dstRgn
On return, a handle to the region holding the resulting union area. If both regions are empty, `UnionRgn` sets the destination to the empty region defined by the rectangle (0,0,0,0).

The `UnionRgn` function does not create the destination region; you must have already allocated memory for it by using the `NewRgn` (page 2726) function.

The destination region may be one of the source regions, if desired.

Discussion
The UnionRgn procedure calculates the union of the two regions whose handles you pass in the srcRgnA and srcRgnB parameters, and it places the union in the region whose handle you pass in the dstRgn parameter. If both regions are empty, UnionRgn sets the destination to the empty region defined by the rectangle (0,0,0,0).

Special Considerations

The `UnionRgn` function may temporarily use heap space that's twice the size of the two input regions.

The `UnionRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
QuickdrawAPI.h

UnlockPixels

Allows the Memory Manager to move the base address for the offscreen pixel map that you specify in the pm parameter. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void UnlockPixels (
   PixMapHandle pm
);
```

Parameters

pm

A handle to an offscreen pixel map. Pass the same handle that you passed previously to the LockPixels function.

Discussion
To ensure the integrity of the data in a pixel image, call LockPixels before drawing into or copying from a pixel map; then, to prevent heap fragmentation, call UnlockPixels as soon as your application finishes drawing to and copying from the offscreen pixel map.

The baseAddr field of the PixMap structure for an offscreen graphics world contains a handle instead of a pointer (which is what the baseAddr field for an onscreen pixel map contains). The LockPixels function dereferences the PixMap handle into a pointer. When you use the UnlockPixels function, the handle is recovered.

You don't need to call UnlockPixels if LockPixels returns FALSE, because LockPixels doesn't lock the memory for a pixel image if that memory has been purged. However, calling UnlockPixels on purged memory does no harm.

Special Considerations

The UnlockPixels function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
QDOffscreen.h

UnlockPortBits

Releases a previously acquired lock on the back buffer for a Carbon window. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
OSErr UnlockPortBits (
   GrafPtr port
);
```

Parameters

port

A window port specified in a previous call to `LockPortBits` (page 2705).

Return Value
A result code. If `noErr`, the corresponding lock is released.

Discussion
For more information about this function, see `LockPortBits` (page 2705).

In Mac OS 9, this function does nothing and returns `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`QuickdrawAPI.h`

UnpackBits

Decompresses a data buffer containing data compressed by `PackBits`. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
void UnpackBits (
   Ptr *srcPtr,
   Ptr *dstPtr,
   short dstBytes
);
```

Parameters

srcPtr

On entry, a pointer to the first byte of a buffer of data to be decompressed. On exit, a pointer to the first byte following the compressed data.

dstPtr

On entry, a pointer to the first byte in which to store decompressed data. On exit, a pointer to the first byte following the decompressed data.

dstBytes

The number of bytes of the data before compression. Use `PackBits` to compress data structures of a fixed size that you can then pass in this parameter to `UnpackBits`, or store with the compressed data the original size of the uncompressed data.

Discussion
Because your application must allocate memory for the source and destination buffers, `UnpackBits` does not move relocatable blocks. Thus, you can call it at interrupt time.

Because `UnpackBits` changes the values of the `srcPtr` and `dstPtr` parameters, you should pass to `UnpackBits` only copies of the pointers to the source and destination buffers. This allows you to access the beginning of the source and destination buffers after `UnpackBits` returns. Also, if the source or destination buffer is stored in an unlocked, relocatable block, this technique prevents `UnpackBits` from changing the value of a master pointer, which would make the original handle invalid.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

UpdateGWorld

Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world. (Deprecated in Mac OS X v10.4. Use Quartz 2D instead; see *Quartz Programming Guide for QuickDraw Developers*.)

```
GWorldFlags UpdateGWorld (
   GWorldPtr *offscreenGWorld,
   short pixelDepth,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

offscreenGWorld
On input, a pointer to an existing offscreen graphics world; upon completion, the pointer to the updated offscreen graphics world.

pixelDepth
The pixel depth of the offscreen world; possible depths are 1, 2, 4, 8, 16, and 32 bits per pixel. If you specify 0 in this parameter, `UpdateGWorld` rescans the device list and uses the depth of the screen with the greatest pixel depth among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you specify 0 in this parameter, `UpdateGWorld` also copies the `GDevice` structure from this device to create an offscreen `GDevice` structure. The `UpdateGWorld` function ignores the value you supply for this parameter if you specify a `GDevice` structure in the `aGDevice` parameter.

boundsRect
The boundary rectangle and port rectangle for the offscreen pixel map. This also becomes the boundary rectangle for the `GDevice` structure, if `NewGWorld` creates one. If you specify 0 in the `pixelDepth` parameter, `NewGWorld` interprets the boundaries in global coordinates, with which it determines which screens intersect the rectangle. (`NewGWorld` then uses the pixel depth, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle.) Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure. If you pass `NULL` in this parameter, `UpdateGWorld` uses the default color table for the pixel depth that you specify in the `pixelDepth` parameter; if you set the `pixelDepth` parameter to 0, `UpdateGWorld` copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. The `UpdateGWorld` function ignores the value you supply for this parameter if you specify a `GDevice` structure in the `aGDevice` parameter.

aGDevice

As an option, a handle to a `GDevice` structure whose pixel depth and color table you want to use for the offscreen graphics world. To use the pixel depth and color table that you specify in the `pixelDepth` and `cTable` parameters, set this parameter to `NULL`.

flags

Options available to your application. You can set a combination of the flags `clipPix`, `stretchPix`, and `ditherPix`. If you don't wish to use any of these flags, specify 0. However, you should pass either `clipPix` or `stretchPix` to ensure that the pixel map is updated to reflect the new color table. See GWorldFlags (page 2863) for a description of the values you can use here.

Return Value

`UpdateGWorld` returns the `gwFlagErr` flag if `UpdateGWorld` was unsuccessful; in this case, the offscreen graphics world is left unchanged. Use the `QDError` function to help you determine why `UpdateGWorld` failed.

Discussion

You should call `UpdateGWorld` after every update event and whenever your windows move or change size.

If the `LockPixels` (page 2704) function reports that the Memory Manager has purged the base address for the offscreen pixel image, use `UpdateGWorld` to reallocate its memory. Then, reconstruct the pixel image or draw directly in a window instead of preparing the image in an offscreen graphics world.

The `UpdateGWorld` function uses the following algorithm when updating the offscreen pixel image:

1. If the color table that you specify in the `cTable` parameter is different from the previous color table, or if the color table associated with the `GDevice` structure that you specify in the `aGDevice` parameter is different, Color QuickDraw maps the pixel values in the offscreen pixel map to the new color table.
2. If the value you specify in the `pixelDepth` parameter differs from the previous pixel depth, Color QuickDraw translates the pixel values in the offscreen pixel image to those for the new pixel depth.
3. If the rectangle you specify in the `boundsRect` parameter differs from, but has the same size as, the previous boundary rectangle, QuickDraw realigns the pixel image to the screen for optimum performance for the `CopyBits` function.
4. If the rectangle you specify in the `boundsRect` parameter is smaller than the previous boundary rectangle and you specify the `clipPix` flag, the pixel image is clipped along the bottom and right edges.
5. If the rectangle you specify in the `boundsRect` parameter is bigger than the previous boundary rectangle and you specify the `clipPix` flag, the bottom and right edges of the pixel image are undefined.
6. If the rectangle you specify in the `boundsRect` parameter is smaller than the previous boundary rectangle and you specify the `stretchPix` flag, the pixel image is reduced to the new size.
7. If the rectangle you specify in the `boundsRect` parameter is bigger than the previous boundary rectangle and you specify the `stretchPix` flag, the pixel image is stretched to the new size.

8. If the Memory Manager purged the base address for the offscreen pixel image, `UpdateGWorld` reallocates the memory, but the pixel image is lost. You must reconstruct it.

Special Considerations

The `UpdateGWorld` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QDOffscreen.h`

XorRgn

Calculates the difference between the union and the intersection of two regions.

```
void XorRgn (
    RgnHandle srcRgnA,
    RgnHandle srcRgnB,
    RgnHandle dstRgn
);
```

Parameters

srcRgnA

A handle to the first of two regions to compare.

srcRgnB

A handle to the second of two regions to compare.

dstRgn

On return, a handle to the region holding the result.

This does not create the destination region; you must have already allocated memory for it by using the `NewRgn` (page 2726) function.

If the regions are coincident, `XorRgn` sets the destination region to the empty region defined by the rectangle (0,0,0,0).

Discussion

The XorRgn procedure calculates the difference between the union and the intersection of the regions whose handles you pass in the srcRgnA and srcRgnB parameters and places the result in the region whose handle you pass in the dstRgn parameter.

Special Considerations

The `XorRgn` function may temporarily use heap space that's twice the size of the two input regions.

The `XorRgn` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`QuickdrawAPI.h`

Callbacks

ColorComplementProcPtr

Defines a pointer to a color inversion callback function that overrides the Color Manager's color inversion method.

```
typedef Boolean (*ColorComplementProcPtr) (
    RGBColor * rgb
);
```

If you name your function `MyColorComplementProc`, you would declare it like this:

```
Boolean ColorComplementProcPtr (
    RGBColor * rgb
);
```

Parameters

rgb

A pointer to the `RGBColor` data structure. Change it to reflect the inverted value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

ColorSearchProcPtr

Defines a pointer to a color search callback function that overrides the Color Manager's code for inverse table mapping.

```
typedef Boolean (*ColorSearchProcPtr) (
    RGBColor * rgb,
    long * position
);
```

If you name your function `MyColorSearchProc`, you would declare it like this:

```
Boolean ColorSearchProcPtr (
    RGBColor * rgb,
    long * position
);
```

Parameters

rgb

A pointer to the `RGBColor` data structure passed to your search function. Your function should set the `ColorSpec.value` field to the index corresponding to the color indicated here.

position

A pointer to the index of the best-mapping color your function finds.

Return Value

`True` if your function succeeds, `false` if your function cannot find a match.

Discussion

Your `MyColorSearchCallback` function should examine the `RGBColor` data structure passed to it by the Color Manager and return the index to the best-mapping color in the current `GDevice` data structure.

The Color Manager specifies the desired color in the `RGBColor` field of a `ColorSpec` data structure and passes it by a pointer on the stack. Your function should return the corresponding index in the `ColorSpec.value` field. If your function cannot handle the search, return `false` as the function value, and pass the `RGBColor` data structure back to the Color Manager in the `rgb` parameter.

The Color Manager calls each search function in the list until one returns the Boolean value `true`. If no search function installed in the linked list returns `true`, the Color Manager calls the default search function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

CustomXFerProcPtr

```
typedef void (*CustomXFerProcPtr) (
    CustomXFerRecPtr info
);
```

If you name your function `MyCustomXFerProc`, you would declare it like this:

```
void CustomXFerProcPtr (
    CustomXFerRecPtr info
);
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawAPI.h`

DeviceLoopDrawingProcPtr

```
typedef void (*DeviceLoopDrawingProcPtr) (
    short depth,
    short deviceFlags,
    GDHandle targetDevice,
    long userData
);
```

If you name your function `MyDeviceLoopDrawingProc`, you would declare it like this:

```
void DeviceLoopDrawingProcPtr (
    short depth,
    short deviceFlags,
    GDHandle targetDevice,
    long userData
);
```

Parameters

depth

The pixel depth of the graphics device.

deviceFlags

Constants which represent bits that are set to 1 in the `gdFlags` field of the `GDevice` structure for the current device. See "Device Attribute Constants" (page 2887) for a description of the values which you can receive in this parameter.

targetDevice

A handle to the `GDevice` (page 2859) structure for the current device.

userData

A value that your application supplies to the `DeviceLoop` function, which in turn passes the value to your drawing function for whatever purpose you deem useful.

Discussion

For each video device that intersects a drawing region that you define (generally, the update region of a window), `DeviceLoop` calls your drawing function. Your drawing function should analyze the pixel depth passed in the `depth` parameter and the values passed in the `deviceFlags` parameter, and then draw in a manner that is optimized for the current device.

When highlighting, for example, your application might invert black and white when drawing onto a 1-bit video device but use magenta as the highlight color when drawing onto a color video device. In this case, even were your window to span both a black-and-white and a color screen, the user sees the selection inverted on the black-and-white screen, while magenta would be used to highlight the selection on the color screen.

You must provide a pointer to your `MyDeviceLoopDrawingCallback` function in the `drawingProc` parameter for `DeviceLoop`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

DragGrayRgnProcPtr

```
typedef void (*DragGrayRgnProcPtr) (
);
```

If you name your function `MyDragGrayRgnProc`, you would declare it like this:

```
void DragGrayRgnProcPtr ();
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDArcProcPtr

```
typedef void (*QDArcProcPtr) (
    GrafVerb verb,
    const Rect * r,
    short startAngle,
    short arcAngle
);
```

If you name your function `MyQDArcProc`, you would declare it like this:

```
void QDArcProcPtr (
    GrafVerb verb,
    const Rect * r,
    short startAngle,
    short arcAngle
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDBitsProcPtr

```
typedef void (*QDBitsProcPtr) (
    const BitMap * srcBits,
    const Rect * srcRect,
    const Rect * dstRect,
    short mode,
    RgnHandle maskRgn
);
```

If you name your function `MyQDBitsProc`, you would declare it like this:

```
void QDBitsProcPtr (
    const BitMap * srcBits,
```

```
    const Rect * srcRect,
    const Rect * dstRect,
    short mode,
    RgnHandle maskRgn
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDCommentProcPtr

```
typedef void (*QDCommentProcPtr) (
    short kind,
    short dataSize,
    Handle dataHandle
);
```

If you name your function `MyQDCommentProc`, you would declare it like this:

```
void QDCommentProcPtr (
    short kind,
    short dataSize,
    Handle dataHandle
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDGetPicProcPtr

```
typedef void (*QDGetPicProcPtr) (
    void * dataPtr,
    short byteCount
);
```

If you name your function `MyQDGetPicProc`, you would declare it like this:

```
void QDGetPicProcPtr (
    void * dataPtr,
    short byteCount
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDJShieldCursorProcPtr

```
typedef void (*QDJShieldCursorProcPtr) (
    short left,
    short top,
    short right,
    short bottom
);
```

If you name your function `MyQDJShieldCursorProc`, you would declare it like this:

```
void QDJShieldCursorProcPtr (
    short left,
    short top,
    short right,
    short bottom
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDLineProcPtr

```
typedef void (*QDLineProcPtr) (
    Point newPt
);
```

If you name your function `MyQDLineProc`, you would declare it like this:

```
void QDLineProcPtr (
    Point newPt
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDOpcodeProcPtr

```
typedef void (*QDOpcodeProcPtr) (
    const Rect * fromRect,
    const Rect * toRect,
    UInt16 opcode,
    SInt16 version
);
```

If you name your function `MyQDOpcodeProc`, you would declare it like this:

```
void QDOpcodeProcPtr (
    const Rect * fromRect,
    const Rect * toRect,
    UInt16 opcode,
    SInt16 version
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDOvalProcPtr

```
typedef void (*QDOvalProcPtr) (
    GrafVerb verb,
    const Rect * r
);
```

If you name your function `MyQDOvalProc`, you would declare it like this:

```
void QDOvalProcPtr (
    GrafVerb verb,
    const Rect * r
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDPolyProcPtr

```
typedef void (*QDPolyProcPtr) (
    GrafVerb verb,
    PolyHandle poly
);
```

If you name your function `MyQDPolyProc`, you would declare it like this:

```
void QDPolyProcPtr (
    GrafVerb verb,
    PolyHandle poly
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDPrinterStatusProcPtr

```
typedef OSStatus (*QDPrinterStatusProcPtr) (
    PrinterStatusOpcode opcode,
    CGrafPtr currentPort,
    void * printerStatus
);
```

If you name your function `MyQDPrinterStatusProc`, you would declare it like this:

```
OSStatus QDPrinterStatusProcPtr (
    PrinterStatusOpcode opcode,
    CGrafPtr currentPort,
    void * printerStatus
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDPutPicProcPtr

```
typedef void (*QDPutPicProcPtr) (
    const void * dataPtr,
    short byteCount
);
```

If you name your function `MyQDPutPicProc`, you would declare it like this:

```
void QDPutPicProcPtr (
    const void * dataPtr,
    short byteCount
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDRectProcPtr

```
typedef void (*QDRectProcPtr) (
    GrafVerb verb,
    const Rect * r
);
```

If you name your function `MyQDRectProc`, you would declare it like this:

```
void QDRectProcPtr (
    GrafVerb verb,
    const Rect * r
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDRgnProcPtr

```
typedef void (*QDRgnProcPtr) (
    GrafVerb verb,
    RgnHandle rgn
);
```

If you name your function `MyQDRgnProc`, you would declare it like this:

```
void QDRgnProcPtr (
    GrafVerb verb,
    RgnHandle rgn
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDRRectProcPtr

```
typedef void (*QDRRectProcPtr) (
    GrafVerb verb,
    const Rect * r,
    short ovalWidth,
    short ovalHeight
);
```

If you name your function `MyQDRRectProc`, you would declare it like this:

```
void QDRRectProcPtr (
    GrafVerb verb,
    const Rect * r,
    short ovalWidth,
    short ovalHeight
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDStdGlyphsProcPtr

```
typedef OSStatus (*QDStdGlyphsProcPtr) (
    void * dataStream,
    ByteCount size
);
```

If you name your function `MyQDStdGlyphsProc`, you would declare it like this:

```
OSStatus QDStdGlyphsProcPtr (
    void * dataStream,
    ByteCount size
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDTextProcPtr

```
typedef void (*QDTextProcPtr) (
    short byteCount,
    const void * textBuf,
    Point numer,
    Point denom
);
```

If you name your function `MyQDTextProc`, you would declare it like this:

```
void QDTextProcPtr (
    short byteCount,
    const void * textBuf,
    Point numer,
    Point denom
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDTxMeasProcPtr

```
typedef short (*QDTxMeasProcPtr) (
    short byteCount,
    const void * textAddr,
    Point * numer,
    Point * denom,
    FontInfo * info
);
```

If you name your function `MyQDTxMeasProc`, you would declare it like this:

```
short QDTxMeasProcPtr (
    short byteCount,
    const void * textAddr,
    Point * numer,
    Point * denom,
    FontInfo * info
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

RegionToRectsProcPtr

```
typedef OSStatus (*RegionToRectsProcPtr) (
    UInt16 message,
    RgnHandle rgn,
    const Rect * rect,
    void * refCon
);
```

If you name your function `MyRegionToRectsProc`, you would declare it like this:

```
OSStatus RegionToRectsProcPtr (
    UInt16 message,
    RgnHandle rgn,
    const Rect * rect,
    void * refCon
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawAPI.h`

Data Types

BitMap

```
struct BitMap {
    Ptr baseAddr;
    short rowBytes;
    Rect bounds;
};
typedef struct BitMap BitMap;
typedef BitMap * BitMapPtr;
```

Fields
`baseAddr`
: A pointer to the beginning of the bit image.

`rowBytes`
: The offset in bytes from one row of the image to the next. The value of the `rowBytes` field must be less than $4000.

`bounds`
: The bitmap's boundary rectangle by default, the entire main screen.

Discussion
A bitmap, which is a data structure of type `BitMap`, defines a bit image in terms of the QuickDraw coordinate plane. (A bit image is a collection of bits in memory that form a grid.)

A bitmap has three parts: a pointer to a bit image, the row width of that image, and a boundary rectangle that links the local coordinate system of a graphics port to QuickDraw's global coordinate system and defines the area of the bit image into which QuickDraw can draw.

The width of the boundary rectangle determines how many bits of one row are logically owned by the bitmap. This width must not exceed the number of bits in each row of the bit image. The height of the boundary rectangle determines how many rows of the image are logically owned by the bitmap. The number of rows enclosed by the boundary rectangle must not exceed the number of rows in the bit image.

The boundary rectangle defines the local coordinate system used by the port rectangle for a graphics port (described next). The upper-left corner (which for a window is called the window origin) of the port rectangle usually has a vertical coordinate of 0 and a horizontal coordinate of 0, although you can use the function `SetOrigin` (page 2797) to change the coordinates of the window origin.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

Bits16

```
typedef short Bits16[16];
```

Discussion
The `Bits16` array is used by the `Cursor` (page 2854) structure to hold a black-and-white, 16-by-16 pixel square image.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

CCrsr

```
struct CCrsr {
    short crsrType;
    PixMapHandle crsrMap;
    Handle crsrData;
    Handle crsrXData;
    short crsrXValid;
    Handle crsrXHandle;
    Bits16 crsr1Data;
    Bits16 crsrMask;
    Point crsrHotSpot;
    long crsrXTable;
    long crsrID;
};
typedef struct CCrsr CCrsr;
typedef CCrsr * CCrsrPtr;
```

Fields

`crsrType`
The type of cursor. Possible values are $8000 for a black-and-white cursor and $8001 for a color cursor.

`crsrMap`
A handle to the `PixMap` structure defining the cursor's characteristics. When the screen depth is greater than 2 bits per pixel, the `crsrMap` field and the `crsrData` field define the image. The pixels within the mask replace the destination pixels. Color QuickDraw transfers the pixels outside the mask into the destination pixels using the XOR Boolean transfer mode. Therefore, if pixels outside the mask are white, the destination pixels aren't changed. If pixels outside the mask are all black, the destination pixels are inverted. All other values outside of the mask cause unpredictable results.

`crsrData`
A handle to the cursor's pixel data. To work properly, a color cursor's image should contain white pixels `(R=G= B=$FFFF)` for the transparent part of the image, and black pixels `(R=G=B=$0000)` for the part of the image to be inverted, in addition to the other colors in the cursor's image. Thus, to define a cursor that contains two colors, it's necessary to use a 2-bit cursor image (that is, a four-color image.

`crsrXData`
A handle to the expanded pixel image used internally by Color QuickDraw.

`crsrXValid`
The depth of the expanded cursor image. If you change the cursor's data or color table, set this field to 0 to cause the cursor to be re-expanded. Never set it to any other values.

`crsrXHandle`
Reserved for future use.

`crsr1Data`
A 16-by-16 pixel image with a pixel depth of 1 to be displayed when the cursor is on screens with pixel depths of 1 or 2 bits.

`crsrMask`
The cursor's mask data. QuickDraw uses the mask to crop the cursor's outline into a background color or pattern. QuickDraw then draws the cursor into this shape. The same 1-bit mask is used with images specified by the `crsrData` and `crsr1Data` fields.

`crsrHotSpot`
The cursor's hot spot.

`crsrXTable`
Reserved for future use.

`crsrID`
The color table seed for the cursor.

Discussion
Your application typically does not create `CCrsr` structures. Although you can create a `CCrsr` structure, it is usually easier to create a color cursor in a color cursor resource, 'crsr'.

A color cursor is a 256-pixel color image in a 16-by-16 pixel square defined in a color cursor (`'crsr'`) resource. When your application uses the `GetCCursor` function to get a color cursor from a `'crsr'` resource, `GetCCursor` loads the resource into memory as a `CCrsr` structure. Your application can then display the color cursor by using the `SetCCursor` (page 2790) function.

`CCrsr` is substantially different from the `Cursor` structure. The fields `crsr1Data`, `crsrMask`, and `crsrHotSpot` in the `CCrsr` structure are the only ones that have counterparts in the `Cursor` structure.

The first four fields of the `CCrsr` structure are similar to the first four fields of the `PixPat` record, and are used in the same manner by QuickDraw.

The display of a cursor involves a relationship between a mask, stored in the `crsrMask` field with the same format used for 1-bit cursor masks, and an image. There are two possible sources for a color cursor's image. When the cursor is on a screen whose depth is 1 or 2 bits per pixel, the image for the cursor is taken from the `crsr1Data` field, which contains bitmap cursor data, similar to the bitmap in a `'CURS'` resource.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

CGrafPort

```
struct CGrafPort {
    SInt16 device;
    PixMapHandle portPixMap;
    SInt16 portVersion;
    Handle grafVars;
    SInt16 chExtra;
    SInt16 pnLocHFrac;
    Rect portRect;
    RgnHandle visRgn;
    RgnHandle clipRgn;
    PixPatHandle bkPixPat;
    RGBColor rgbFgColor;
    RGBColor rgbBkColor;
    Point pnLoc;
    Point pnSize;
    SInt16 pnMode;
    PixPatHandle pnPixPat;
    PixPatHandle fillPixPat;
    SInt16 pnVis;
    SInt16 txFont;
    StyleField txFace;
    SInt16 txMode;
    SInt16 txSize;
    Fixed spExtra;
    SInt32 fgColor;
    SInt32 bkColor;
    SInt16 colrBit;
    SInt16 patStretch;
    Handle picSave;
    Handle rgnSave;
    Handle polySave;
    CQDProcsPtr grafProcs;
};
```

CGrafPtr

```
typedef GrafPtr CGrafPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

ColorComplementUPP

```
typedef ColorComplementProcPtr ColorComplementUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

ColorSearchUPP

```
typedef ColorSearchProcPtr ColorSearchUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

ColorSpec

```
struct ColorSpec {
    short value;
    RGBColor rgb;
};
typedef struct ColorSpec ColorSpec;
typedef ColorSpec * ColorSpecPtr;
```

Fields

`value`

The pixel value assigned by QuickDraw for the color specified in the `rgb` field of this structure. QuickDraw assigns a pixel value based on the capabilities of the user's screen. For indexed devices, the pixel value is an index number assigned by QuickDraw to the closest color available on the indexed device for direct devices, this value expresses the best available red, green, and blue values for the color on the direct device.

`rgb`

An `RGBColor` (page 2882) structure that fully specifies the color whose approximation QuickDraw specifies in the `value` field.

Discussion
When creating a `PixMap` (page 2869) structure for an indexed device, QuickDraw creates a `ColorTable` structure that defines the best colors available for the pixel image on that graphics device. QuickDraw also stores a `ColorTable` structure for the currently available colors in the graphics device's CLUT.

One of the fields in a `ColorTable` structure requires a value of type `cSpecArray`, which is defined as an array of `ColorSpec` structures. Typically, your application never needs to create `ColorTable` structures or `ColorSpec` structures. For completeness, the data structure of type `ColorSpec` is shown here.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSTypes.h`

ColorTable

```
struct ColorTable {
    long ctSeed;
    short ctFlags;
    short ctSize;
    CSpecArray ctTable;
};
typedef struct ColorTable ColorTable;
typedef ColorTable * CTabPtr;
typedef CTabPtr * CTabHandle;
```

Fields

`ctSeed`

Identifies a particular instance of a color table. QuickDraw uses the `ctSeed` value to compare an indexed device's color table with its associated inverse table (a table it uses for fast color lookup). When the color table for a graphics device has been changed, QuickDraw needs to rebuild the inverse table.

`ctFlags`

Flags that distinguish pixel map color tables from color tables in `GDevice` structures.

`ctSize`

One less than the number of entries in the table.

`ctTable`

An array of `ColorSpec` (page 2850) entries, each containing a pixel value and a color specified by an `RGBColor` structure.

Discussion

When creating a `PixMap` (page 2869) structure for a particular graphics device, QuickDraw creates a `ColorTable` structure that defines the best colors available for the pixel image on that particular graphics device. QuickDraw also creates a `ColorTable` structure of all available colors for use by the CLUT on indexed devices.

Typically, your application needs to create `ColorTable` structures only if it uses the Palette Manager.

Your application should never need to directly change the fields of a `ColorTable` structure. If you find it absolutely necessary for your application to do so, immediately use the `CTabChanged` (page 2593) function to notify QuickDraw that your application has changed the `ColorTable` structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

ConstPatternParam

```
typedef const Pattern* ConstPatternParam;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

CProcRec

```
struct CProcRec {
    Handle nxtComp;
    ColorComplementUPP compProc;
};
typedef struct CProcRec CProcRec;
typedef CProcRec * CProcPtr;
```

Fields

`nxtComp`

A handle to the next `CPRocRec` data structure in the list.

`compProc`

A pointer to a complement function, as described in `ColorComplementProcPtr` (page 2834).

Discussion

The `CProcRec` data structure contains a pointer to a custom complement function and a pointer to the next complement function in the list.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

CQDProcs

```
struct CQDProcs {
    QDTextUPP textProc;
    QDLineUPP lineProc;
    QDRectUPP rectProc;
    QDRRectUPP rRectProc;
    QDOvalUPP ovalProc;
    QDArcUPP arcProc;
    QDPolyUPP polyProc;
    QDRgnUPP rgnProc;
    QDBitsUPP bitsProc;
    QDCommentUPP commentProc;
    QDTxMeasUPP txMeasProc;
    QDGetPicUPP getPicProc;
    QDPutPicUPP putPicProc;
    QDOpcodeUPP opcodeProc;
    UniversalProcPtr newProc1;
    QDStdGlyphsUPP glyphsProc;
    QDPrinterStatusUPP printerStatusProc;
    UniversalProcPtr newProc4;
    UniversalProcPtr newProc5;
    UniversalProcPtr newProc6;
};
typedef struct CQDProcs CQDProcs;
typedef CQDProcs * CQDProcsPtr;
```

Fields

`textProc`

A pointer to the low-level function that draws text. The standard QuickDraw function is the `StdText` function.

`lineProc`
: A pointer to the low-level function that draws lines. The standard QuickDraw function is the `StdLine` function.

`rectProc`
: A pointer to the low-level function that draws rectangles. The standard QuickDraw function is the `StdRect` function.

`rRectProc`
: A pointer to the low-level function that draws rounded rectangles. The standard QuickDraw function is the `StdRRect` function.

`ovalProc`
: A pointer to the low-level function that draws ovals. The standard QuickDraw function is the `StdOval` function.

`arcProc`
: A pointer to the low-level function that draws arcs. The standard QuickDraw function is the `StdArc` function.

`polyProc`
: A pointer to the low-level function that draws polygons. The standard QuickDraw function is the `StdPoly` function.

`rgnProc`
: A pointer to the low-level function that draws regions. The standard QuickDraw function is the `StdRgn` function.

`bitsProc`
: A pointer to the low-level function that copies bitmaps. The standard QuickDraw function is the `StdBits` function.

`commentProc`
: A pointer to the low-level function for processing a picture comment. The standard QuickDraw function is the `StdComment` function.

`txMeasProc`
: A pointer to the low-level function for measuring text width. The standard QuickDraw function is the `StdTxMeas` function.

`getPicProc`
: A pointer to the low-level function for retrieving information from the definition of a picture. The standard QuickDraw function is the `StdGetPic` function.

`putPicProc`
: A pointer to the low-level function for saving information as the definition of a picture. The standard QuickDraw function is the `StdPutPic` function.

`opcodeProc`
: Reserved for future use.

`newProc1`
: Reserved for future use.

`glyphsProc`
: Reserved for future use.

`printerStatusProc`
: Reserved for future use.

`newProc4`
: Reserved for future use.

`newProc5`
Reserved for future use.

`newProc6`
Reserved for future use.

Discussion
Use the `CQDProcs` structure only if you customize one or more of QuickDraw's standard low-level drawing functions. Use the `SetStdCProcs` (page 2811) function to create a `CQDProcs` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

CSpecArray

```
typedef ColorSpec CSpecArray[1];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

Cursor

```
struct Cursor {
    Bits16 data;
    Bits16 mask;
    Point hotSpot;
};
typedef struct Cursor Cursor;
typedef Cursor * CursPtr;
```

Fields
`data`
Cursor image data, which must begin on a word boundary.

`mask`
The cursor's mask. QuickDraw uses the mask to crop the cursor's outline into a background color or pattern. QuickDraw then draws the cursor into this shape.

`hotSpot`
A point in the image that aligns a point (not a bit) in the image with the mouse location on the screen. Whenever the user moves the mouse, the low-level interrupt-driven mouse functions move the cursor. When the user clicks, the Event Manager function `WaitNextEvent` reports the location of the cursor's hot spot in global coordinates.

Discussion
Your application typically does not create `Cursor` structures. Although you can create a `Cursor` structure and its associated `Bits16` array in your program code, it is usually easier to create a black-and-white cursor in a cursor resource,'CURS'.

A cursor is a 256-pixel, black-and-white image in a 16-by-16 pixel square. When your application uses the `GetCursor` (page 2638) function to get a cursor from a `'CURS'` resource, `GetCursor` loads the resource into memory as a `Cursor` structure. Your application then displays the color cursor by using the `SetCursor` (page 2792) function.

The cursor appears on the screen as a 16-by-16 pixel square. The appearance of each bit if the square is determined by the corresponding bits in the data and the mask and, if the mask bit is 0, by the pixel under the cursor. The four possible combinations of values for the data bit and the mask bit are:

- Data bit 0, Mask bit 1. The resulting pixel on the screen is white.
- Data bit 1, Mask bit 1. The resulting pixel on the screen is black.
- Data bit 0, Mask bit 0. The resulting pixel on the screen is the same as the pixel under the cursor.
- Data bit 1, Mask bit 0. The resulting pixel on the screen is the inverse of the pixel under the cursor.

Notice that if all mask bits are 0, the cursor is completely transparent, in that the image under the cursor can still be viewed. Pixels under the white part of the cursor appear unchanged; under the black part of the cursor, black pixels show through as white.

Basic QuickDraw supplies a predefined cursor in the global variable named `arrow`; this is the standard arrow cursor.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`X.h`

CursorImageRec

```
struct CursorImageRec {
    UInt16 majorVersion;
    UInt16 minorVersion;
    PixMapHandle cursorPixMap;
    BitMapHandle cursorBitMask;
};
typedef struct CursorImageRec CursorImageRec;
typedef CursorImageRec * CursorImagePtr;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`Quickdraw.h`

CursorInfo

```
struct CursorInfo {
    long version;
    long capabilities;
    long animateDuration;
    Rect bounds;
    Point hotspot;
    long reserved;
};
typedef struct CursorInfo CursorInfo;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawAPI.h

CustomXFerRec

```
struct CustomXFerRec {
    UInt32 version;
    void * srcPixels;
    void * destPixels;
    void * resultPixels;
    UInt32 refCon;
    UInt32 pixelSize;
    UInt32 pixelCount;
    Point firstPixelHV;
    Rect destBounds;
};
typedef struct CustomXFerRec CustomXFerRec;
typedef CustomXFerRec * CustomXFerRecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawAPI.h

CWindowPtr

```
typedef WindowPtr CWindowPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

DeviceLoopDrawingUPP

```
typedef DeviceLoopDrawingProcPtr DeviceLoopDrawingUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

DeviceLoopFlags

```
typedef unsigned long DeviceLoopFlags;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

DialogPtr

An opaque type that represents a dialog.

```
typedef struct OpaqueDialogPtr * DialogPtr;
```

Discussion
This is a Dialog Manager data type, defined in QuickDraw for historical reasons. Its role in Mac OS X is to serve as the basis for the widely used `DialogRef` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

DragConstraint

```
typedef UInt16 DragConstraint;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

DragGrayRgnUPP

```
typedef DragGrayRgnProcPtr DragGrayRgnUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

GammaTbl

```
struct GammaTbl {
    short gVersion;
    short gType;
    short gFormulaSize;
    short gChanCnt;
    short gDataCnt;
    short gDataWidth;
    short gFormulaData[1];
};
typedef struct GammaTbl GammaTbl;
typedef GammaTbl * GammaTblPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSTypes.h`

GDevice

```
struct GDevice {
    short gdRefNum;
    short gdID;
    short gdType;
    ITabHandle gdITable;
    short gdResPref;
    SProcHndl gdSearchProc;
    CProcHndl gdCompProc;
    short gdFlags;
    PixMapHandle gdPMap;
    long gdRefCon;
    GDHandle gdNextGD;
    Rect gdRect;
    long gdMode;
    short gdCCBytes;
    short gdCCDepth;
    Handle gdCCXData;
    Handle gdCCXMask;
    long gdReserved;
};
typedef struct GDevice GDevice;
typedef GDevice * GDPtr;
typedef GDPtr * GDHandle;
```

Fields

`gdRefNum`
The reference number of the driver for the screen associated with the video device. For most video devices, this information is set at system startup time.

`gdID`
Reserved. If you create your own `GDevice` structure, set this field to 0.

`gdType`
The general type of graphics device. See "Graphics Device Type Constants" (page 2891) for a description of the values which you can use in this field.

`gdITable`
A handle to the inverse table for color mapping.

`gdResPref`
The preferred resolution for inverse tables.

`gdSearchProc`
A handle to the list of search functions. Its value is `NULL` for the default function.

`gdCompProc`
A handle to a list of complement functions. Its value is `NULL` for the default function.

`gdFlags`
The `GDevice` structure's attributes. To set the attribute bits in the `gdFlags` field, use the `SetDeviceAttribute` (page 2793) function. Do not set `gdFlags` directly in the `GDevice` structure.

`gdPMap`
: A handle to a `PixMap` structure giving the dimension of the image buffer, along with the characteristics of the graphics device (resolution, storage format, color depth, and color table). For `GDevice` structures, the high bit of the global variable

```
(((**TheGDevice).**gdPMap).**pmTable).ctFlags
```

is always set.

`gdRefCon`
: A value used by system software to pass device-related parameters. Since a graphics device is shared, do not store data here.

`gdNextGD`
: A handle to the next graphics device in the device list. If this is the last graphics device in the device list, the field contains 0.

`gdRect`
: The boundary rectangle of the graphics device represented by the `GDevice` structure. The main screen has the upper-left corner of the rectangle set to (0,0). All other graphics devices are relative to this point.

`gdMode`
: The current setting for the graphics device mode. This value is passed to the video driver to set its pixel depth and to specify color or black and white; applications do not need this information.

`gdCCBytes`
: The `rowBytes` value of the expanded cursor. Your application should not change this field.

`gdCCDepth`
: The depth of the expanded cursor. Your application should not change this field.

`gdCCXData`
: A handle to the cursor's expanded data. Your application should not change this field.

`gdCCXMask`
: A handle to the cursor's expanded mask. Your application should not change this field.

`gdReserved`
: Reserved for future expansion; it must be set to 0 for future compatibility.

Discussion

Color QuickDraw stores state information for video devices and offscreen graphics worlds in `GDevice` structures. When the system starts up, it allocates and initializes one handle to a `GDevice` structure for each video device it finds. When you use the Offscreen Graphics Devices function, `NewGWorld`, Color QuickDraw automatically creates a `GDevice` structure for the new offscreen graphics world. The system links these `GDevice` structures in a list, called the device list. (You can find a handle to the first element in the device list in the global variable `DeviceList`.) By default, the `GDevice` structure corresponding to the first video device found is marked as the current device. All other graphics devices in the list are initially marked as inactive.

When the user moves a window or creates a window on another screen, and your application draws into that window, Color QuickDraw automatically makes the video device for that screen the current device. Color QuickDraw stores that information in the global variable `TheGDevice`.

`GDevice` structures that correspond to video devices have drivers associated with them. These drivers can be used to change the mode of the video device from black and white to color and to change the pixel depth. Application-created `GDevice` structures usually don't require drivers.

Your application should never need to directly change the fields of a `GDevice` structure. If you find it absolutely necessary for your application to so, immediately use the `GDeviceChanged` function to notify QuickDraw that your application has changed the `GDevice` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

GrafPort

```
struct GrafPort {
    SInt16 device;
    BitMap portBits;
    Rect portRect;
    RgnHandle visRgn;
    RgnHandle clipRgn;
    Pattern bkPat;
    Pattern fillPat;
    Point pnLoc;
    Point pnSize;
    SInt16 pnMode;
    Pattern pnPat;
    SInt16 pnVis;
    SInt16 txFont;
    StyleField txFace;
    SInt16 txMode;
    SInt16 txSize;
    Fixed spExtra;
    SInt32 fgColor;
    SInt32 bkColor;
    SInt16 colrBit;
    SInt16 patStretch;
    Handle picSave;
    Handle rgnSave;
    Handle polySave;
    QDProcsPtr grafProcs;
};
```

GrafPtr

```
typedef struct OpaqueGrafPtr * GrafPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

GrafVars

```
struct GrafVars {
    RGBColor rgbOpColor;
    RGBColor rgbHiliteColor;
    Handle pmFgColor;
    short pmFgIndex;
    Handle pmBkColor;
    short pmBkIndex;
    short pmFlags;
};
typedef struct GrafVars GrafVars;
typedef GrafVars * GVarPtr;
```

Fields

`rgbOpColor`
The color for the arithmetic transfer operations `addPin`, `subPin`, and `blend`.

`rgbHiliteColor`
The highlight color for this graphics port.

`pmFgColor`
A handle to the palette that contains the foreground color.

`pmFgIndex`
The index value into the palette for the foreground color.

`pmBkColor`
A handle to the palette that contains the background color.

`pmBkIndex`
The index value into the palette for the background color.

`pmFlags`
Flags private to the Palette Manager.

Discussion

The `GrafVars` structure contains color information in addition to that in the `CGrafPort` structure, of which it is logically a part; the information is used by QuickDraw and the Palette Manager.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

GrafVerb

```
typedef SInt8 GrafVerb;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

GWorldFlags

```
typedef unsigned long GWorldFlags;
```

Discussion
Several functions expect or return values defined by the `GWorldFlags` data type. See "Graphics World Flags" (page 2891) for a detailed description of these flags.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QDOffscreen.h`

GWorldPtr

Defines a pointer to a structure that your application can use to refer to an offscreen graphics world.

```
typedef CGrafPtr GWorldPtr;
```

Discussion
An offscreen graphics world in color QuickDraw contains a `CGrafPort` structure—and its handles to associated `PixMap` and `ColorTable` structures—that describes an offscreen graphics port and contains references to a `GDevice` structure and other state information. The actual data structure for an offscreen graphics world is kept private to allow for future extensions. However, when your application uses the `NewGWorld` function to create an offscreen world, `NewGWorld` returns a pointer of type `GWorldPtr` by which your application refers to the offscreen graphics world.

On computers lacking color QuickDraw, `GWorldPtr` points to an extension of the `GrafPort` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QDOffscreen.h`

ITab

```
struct ITab {
    long iTabSeed;
    short iTabRes;
    Byte iTTable[1];
};
typedef struct ITab ITab;
typedef ITab * ITabPtr;
typedef ITabPtr * ITabHandle;
```

Fields

`iTabSeed`
The `iTabSeed` value, initially set from the corresponding CLUT's `ctSeed` field. If at any time these do not match, then the color table was changed, and the inverse table needs to be rebuilt.

`iTabRes`
The resolution of this inverse table.

`iTTable`
An array of index values. The size of the `iTabTable` field in bytes is `23*iTabRes`.

Discussion
The `ITab` data structure contains the inverse table information that the Color Manager uses for fast mapping of RGB color values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

MacPolygon

```
struct MacPolygon {
    short polySize;
    Rect polyBBox;
    Point polyPoints[1];
};
typedef struct MacPolygon MacPolygon;
typedef MacPolygon Polygon;
typedef MacPolygon * PolyPtr;
typedef PolyPtr * PolyHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

MacRegion

```
struct MacRegion {
    UInt16 rgnSize;
    Rect rgnBBox;
};
typedef struct MacRegion MacRegion;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

MatchRec

```
struct MatchRec {
    unsigned short red;
    unsigned short green;
    unsigned short blue;
    long matchData;
};
typedef struct MatchRec MatchRec;
```

Fields

`red`
Red value of the seed.

`green`
Green value of the seed.

`blue`
Blue value of the seed.

`matchData`
The value passed in the `matchData` parameter of the `SeedCFill` or `CalcCMask` function.

Discussion

When `SeedCFill` (page 2787) or `CalcCMask` (page 2577) calls your color search function, the `GDRefCon` field of the current `GDevice` structure contains a pointer to a `MatchRec` structure. This structure contains the RGB value of the seed pixel or seed color for which your color search function searches.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

OpenCPicParams

```
struct OpenCPicParams {
    Rect srcRect;
    Fixed hRes;
    Fixed vRes;
    short version;
    short reserved1;
    long reserved2;
};
typedef struct OpenCPicParams OpenCPicParams;
```

Fields

`srcRect`
The optimal bounding rectangle for the resolution indicated by the `hRes` and `vRes` fields. When you later call the `DrawPicture` (page 2610) function to play back the saved picture, specify a destination rectangle and `DrawPicture` scales the picture so that it is completely aligned with the destination rectangle.

`hRes`
The best horizontal resolution for the picture. A value of $00480000 specifies a horizontal resolution of 72 dpi.

`vRes`
: The best vertical resolution for the picture. A value of $00480000 specifies a vertical resolution of 72 dpi.

`version`
: Always set this field to –2.

`reserved1`
: Reserved; set to 0.

`reserved2`
: Reserved; set to 0.

Discussion
When you use the `OpenCPicture` function to begin creating a picture, you must pass it information in an `OpenCPicParams` structure. This structure provides a simple mechanism for specifying resolutions when creating images. For example, applications that create pictures from scanned images can specify resolutions higher than 72 dpi for these pictures in `OpenCPicParams` structures.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

Pattern

```
struct Pattern {
    UInt8 pat[8];
};
typedef struct Pattern Pattern;
typedef Pattern * PatPtr;
typedef PatPtr * PatHandle;
```

Discussion
Your application typically does not create `Pattern` structures. Although you can create `Pattern` structures in your program code, it is usually easier to create bit patterns using the pattern, 'PAT', or pattern list, 'PAT#', resource.

A bit pattern is a 64-bit image, organized as an 8-by-8 bit square, that defines a repeating design or tone. When a pattern is drawn, it is aligned so that adjacent areas of the same pattern in the same graphics port form a continuous, coordinated pattern. QuickDraw provides predefined patterns in global variables named `white`, `black`, `gray`, `ltGray`, and `dkGray`. The row width of a pattern is 1 byte.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

PenState

```
struct PenState {
    Point pnLoc;
    Point pnSize;
    short pnMode;
    Pattern pnPat;
};
typedef struct PenState PenState;
```

Fields

`pnLoc`

For the current graphics port at the time the `GetPenState` function was called, the value of that graphics port's `pnLoc` field. This value is the point where QuickDraw begins drawing next. The location of the graphics pen is a point in the graphics port's coordinate system, not a pixel in a bit image. The upper-left corner of the pen is at the pen location the graphics pen hangs below and to the right of this point.

`pnSize`

For the current graphics port at the time the `GetPenState` function was called, the value of that graphics port's `pnSize` field. The graphics pen is rectangular in shape, and its width and height are specified by the values in the `pnSize` field. The default size is a 1-by-1 bit square; the width and height can range from 0 by 0 to 32,767 by 32,767. If either the pen width or the pen height is 0, the pen does not draw. Heights or widths of less than 0 are undefined.

`pnMode`

The pattern mode—that is, for the current graphics port at the time the `GetPenState` function was called, the value of that graphics port's `pnMode` field. This value determines how the pen pattern is to affect what's already in the bit image when lines or shapes are drawn. When the graphics pen draws, QuickDraw first determines what bits in the bit image are affected, finds their corresponding bits in the pattern, and then transfers the bits from the pattern into the image according to this mode, which specifies one of eight Boolean transfer operations. The resulting bit is stored into its proper place in the bit image.

`pnPat`

For the current graphics port at the time the `GetPenState` function was called, the pen pattern for that graphics port. This pattern determines how the bits under the graphics pen are affected when lines or shapes are drawn.

Discussion

The `GetPenState` (page 2647) function saves the location, size, pattern, and pattern mode of the graphics pen for the current graphics port in a `PenState` structure, which is a data structure of type `PenState`. After changing the graphics pen as necessary, you can later restore these pen states with the `SetPenState` (page 2798) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

Picture

```
struct Picture {
    short picSize;
    Rect picFrame;
};
typedef struct Picture Picture;
typedef Picture * PicPtr;
typedef PicPtr * PicHandle;
```

Fields

`picSize`

The size of the rest of this structure for a version 1 picture. To maintain compatibility with the version 1 picture format, the `picSize` field was not changed for the version 2 picture or extended version 2 formats. The information in this field is used only for version 1 pictures, which cannot exceed 32 KB in size. Because version 2 and extended version 2 pictures can be much larger than the 32 KB limit imposed by the 2-byte `picSize` field, you should use the Memory Manager function `GetHandleSize` to determine the size of a picture in memory; you should use the File Manager function `PBGetFInfo` to determine the size of a picture in a `'PICT'` file; and you should use the Resource Manager function `GetMaxResourceSize` to determine the size of a `'PICT'` resource.

`picFrame`

The bounding rectangle for the picture defined in the rest of this structure. The `DrawPicture` function uses this rectangle to scale the picture if you draw it into a destination rectangle of a different size.

Discussion

When you use the `OpenCPicture` (page 2734) or `OpenPicture` (page 2736) function, QuickDraw begins collecting your subsequent drawing commands in a `Picture` structure. (You use the `ClosePicture` function to complete a picture definition.) When you use the `GetPicture` (page 2648) function to retrieve a picture stored in a resource, `GetPicture` reads the resource into memory as a `Picture` structure. By using the `DrawPicture` (page 2610) procedure, you can draw onscreen the picture defined by the commands stored in the `Picture` structure.

A picture opcode is a number that the `DrawPicture` function uses to determine what object to draw or what mode to change for subsequent drawing. Generally, do not read or write this picture data directly. Instead, use the `OpenCPicture` (or `OpenPicture`), `ClosePicture`, and `DrawPicture` functions to process these opcodes.

The `Picture` structure can also contain picture comments. Created by applications using the `PicComment` function, picture comments contain data or commands for special processing by output devices, such as PostScript printers.

You can use File Manager functions to save the picture in a file of type `'PICT'`, you can use Resource Manager functions to save the picture in a resource of type `'PICT'`, and you can use the Scrap Manager function `PutScrap` to store the picture in `'PICT'` scrap format.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

PixelType

```
typedef SInt8 PixelType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

PixMap

```
struct PixMap {
    Ptr baseAddr;
    short rowBytes;
    Rect bounds;
    short pmVersion;
    short packType;
    long packSize;
    Fixed hRes;
    Fixed vRes;
    short pixelType;
    short pixelSize;
    short cmpCount;
    short cmpSize;
    long planeBytes;
    CTabHandle pmTable;
    long pmReserved;
};
typedef struct PixMap PixMap;
typedef PixMap * PixMapPtr;
typedef PixMapPtr * PixMapHandle;
```

Fields

`baseAddr`

For an onscreen pixel image, a pointer to the first byte of the image. For optimal performance, this should be a multiple of 4. The pixel image that appears on a screen is normally stored on a graphics card rather than in main memory.

Note that the `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle instead of a pointer. You must use the `GetPixBaseAddr` function to obtain a pointer to the `PixMap` structure for an offscreen graphics world. Your application should never directly access the `baseAddr` field of the `PixMap` structure for an offscreen graphics world.

`rowBytes`

The offset in bytes from one row of the image to the next. The value must be even, less than $4000, and for best performance it should be a multiple of 4. The high 2 bits of `rowBytes` are used as flags. If bit 15 = 1, the data structure pointed to is a `PixMap` structure; otherwise it is a `BitMap` structure.

`bounds`

The boundary rectangle, which links the local coordinate system of a graphics port to QuickDraw's global coordinate system and defines the area of the bit image into which QuickDraw can draw. By default, the boundary rectangle is the entire main screen. Do not use the value of this field to determine the size of the screen instead use the value of the `gdRect` field of the `GDevice` structure for the screen.

`pmVersion`
: The version number of QuickDraw that created this `PixMap` structure. The value of `pmVersion` is normally 0. If `pmVersion` is 4, QuickDraw treats the `PixMap` structure's `baseAddr` field as 32-bit clean. All other flags are private. Most applications never need to set this field.

`packType`
: The packing algorithm used to compress image data. QuickDraw currently supports a `packType` of 0, which means no packing, and values of 1 to 4 for packing direct pixels.

`packSize`
: The size of the packed image in bytes. When the `packType` field contains the value 0, this field is always set to 0.

`hRes`
: The horizontal resolution of the pixel image in pixels per inch. This value is of type `Fixed`; by default, the value here is $00480000 (for 72 pixels per inch).

`vRes`
: The vertical resolution of the pixel image in pixels per inch. This value is of type `Fixed`; by default, the value here is $00480000 (for 72 pixels per inch).

`pixelType`
: The storage format for a pixel image. Indexed pixels are indicated by a value of 0. Direct pixels are specified by a value of `RGBDirect`, or 16. In the `PixMap` structure of the `GDevice` structure for a direct device, this field is set to the constant `RGBDirect` when the screen depth is set.

`pixelSize`
: Pixel depth; that is, the number of bits used to represent a pixel. Indexed pixels can have sizes of 1, 2, 4, and 8 bits; direct pixel sizes are 16 and 32 bits.

`cmpCount`
: The number of components used to represent a color for a pixel. With indexed pixels, each pixel is a single value representing an index in a color table, and therefore this field contains the value 1—the index is the single component. With direct pixels, each pixel contains three components—one integer each for the intensities of red, green, and blue—so this field contains the value 3.

`cmpSize`
: The size in bits of each component for a pixel. QuickDraw expects that the sizes of all components are the same, and that the value of the `cmpCount` field multiplied by the value of the `cmpSize` field is less than or equal to the value in the `pixelSize` field.

  For an indexed pixel value, which has only one component, the value of the `cmpSize` field is the same as the value of the `pixelSize` field—that is, 1, 2, 4, or 8.

  For direct pixels there are two additional possibilities:

  - A 16-bit pixel, which has three components, has a `cmpSize` value of 5. This leaves an unused high-order bit, which QuickDraw sets to 0.
  - A 32-bit pixel, which has three components (red, green, and blue), has a `cmpSize` value of 8. This leaves an unused high-order byte, which QuickDraw sets to 0.

  Generally, therefore, your application should clear the memory for the image to 0 before creating a 16-bit or 32-bit image. The Memory Manager functions `NewHandleClear` and `NewPtrClear` assist you in allocating pre-zeroed memory.

`planeBytes`
: The offset in bytes from one drawing plane to the next. This field is set to 0.

`pmTable`
: A handle to a `ColorTable` structure for the colors in this pixel map.

`pmReserved`
Reserved for future expansion. This field must be set to 0 for future compatibility.

Discussion
The `PixMap` structure contains information about the dimensions, contents, storage format, depth, resolution, and color usage of a pixel image. The pixel map for a window's color graphics port always consists of the pixel depth, color table, and boundary rectangle of the main screen, even if the window is created on or moved to an entirely different screen.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

PixPat

```
struct PixPat {
    short patType;
    PixMapHandle patMap;
    Handle patData;
    Handle patXData;
    short patXValid;
    Handle patXMap;
    Pattern pat1Data;
};
typedef struct PixPat PixPat;
typedef PixPat * PixPatPtr;
typedef PixPatPtr * PixPatHandle;
```

Fields
`patType`
The pattern's type. The value 0 specifies a basic QuickDraw bit pattern, the value 1 specifies a full-color pixel pattern, and the value 2 specifies an RGB pattern.

`patMap`
A handle to a `PixMap` (page 2869) structure that describes the pattern's pixel image. The `PixMap` structure can contain indexed or direct pixels.

`patData`
A handle to the pattern's pixel image.

`patXData`
A handle to an expanded pixel image used internally by QuickDraw.

`patXValid`
A flag that, when set to –1, invalidates the expanded data.

`patXMap`
Reserved for use by QuickDraw.

`pat1Data`
A bit pattern to be used when this pattern is drawn into a `GrafPort` structure. The `NewPixPat` (page 2720) function sets this field to 50 percent gray.

Discussion
Your application typically does not create `PixPat` structures. Although you can create such structures in your program code, it is usually easier to create pixel patterns using the pixel pattern resource, `'ppat'`.

When used for a color graphics port, the basic QuickDraw functions `PenPat` and `BackPat` store pixel patterns in, respectively, the `pnPixPat` and `bkPixPat` fields of the `CGrafPort` structure and set the `patType` field of the `PixPat` field to 0 to indicate that the `PixPat` structure contains a bit pattern. Such patterns are limited to 8-by-8 pixel dimensions and, instead of being drawn in black and white, are always drawn using the colors specified in the `CGrafPort` structure's `rgbFgColor` and `rgbBkColor` fields, respectively.

In a full-color pixel pattern, the `patType` field contains the value 1, and the pattern's dimensions, depth, resolution, set of colors, and other characteristics are defined by a `PixMap` structure, referenced by the handle in the `patMap` field of the `PixPat` structure. Full-color pixel patterns contain color tables that describe the colors they use. Generally such a color table contains one entry for each color used in the pattern. For instance, if your pattern has five colors, you would probably create a 4 bits per pixel pattern that uses pixel values 0–4, and a color table with five entries, numbered 0–4, that contain the RGB specifications for those pixel values.

However, if you don't specify a color table for a pixel value, QuickDraw assigns a color to that pixel value. The largest unassigned pixel value becomes the foreground color the smallest unassigned pixel value is assigned the background color. Remaining unassigned pixel values are given colors that are evenly distributed between the foreground and background.

For instance, in the color table mentioned above, pixel values 5–15 are unused. Assume that the foreground color is black and the background color is white. Pixel value 15 is assigned the foreground color, black pixel value 5 is assigned the background color, white the nine pixel values between them are assigned evenly distributed shades of gray. If the `PixMap` structure's color table is set to `NULL`, all pixel values are determined by blending the foreground and background colors.

Full-color pixel patterns are not limited to a fixed size: their height and width can be any power of 2, as specified by the height and width of the boundary rectangle for the `PixMap` structure specified in the `patMap` field. A pattern 8 bits wide, which is the size of a bit pattern, has a row width of just 1 byte, contrary to the usual rule that the `rowBytes` field must be even. Read this pattern type into memory using the `GetPixPat` (page 2651) function, and set it using the `PenPixPat` (page 2747) or `BackPixPat` (page 2576) functions.

The pixel map specified in the `patMap` field of the `PixPat` structure defines the pattern's characteristics. The `baseAddr` field of the `PixMap` structure for that pixel map is ignored. For a full-color pixel pattern, the actual pixel image defining the pattern is stored in the handle in the `patData` field of the `PixPat` structure. The pattern's pixel depth need not match that of the pixel map into which it's transferred the depth is adjusted automatically when the pattern is drawn. QuickDraw maintains a private copy of the pattern's pixel image, expanded to the current screen depth and aligned to the current graphics port, in the `patXData` field of the `PixPat` structure.

In an RGB pixel pattern, the `patType` field contains the value 2. Using the `MakeRGBPat` (page 2706) function, your application can specify the exact color it wants to use. QuickDraw selects a pattern to approximate that color. In this way, your application can effectively increase the color resolution of the screen. RGB pixel patterns are particularly useful for dithering: mixing existing colors together to create the illusion of a third color that's unavailable on an indexed device. The `MakeRGBPat` function aids in this process by constructing a dithered pattern to approximate a given absolute color. An RGB pixel pattern can display 125 different patterns on a 4-bit screen, or 2197 different patterns on an 8-bit screen.

An RGB pixel pattern has an 8-by-8 pixel pattern that is 2 bits deep. For an RGB pixel pattern, the `RGBColor` structure that you specify to the `MakeRGBPat` function defines the image; there is no image data.

Your application should never need to directly change the fields of a `PixPat` structure. If you find it absolutely necessary for your application to so, immediately use the `PixPatChanged` (page 2750) function to notify QuickDraw that your application has changed the `PixPat` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

Polygon

```
typedef MacPolygon Polygon;
```

Discussion
After you use the `OpenPoly` function to create a polygon, QuickDraw begins collecting the line-drawing information you provide into a `MacPolygon` structure. The `OpenPoly` function returns a handle to the newly allocated `MacPolygon` structure. Thereafter, your application normally refers to your new polygon by this handle, because QuickDraw functions such as `FramePoly` and `PaintPoly` expect a handle to a `Polygon` as their first parameter.

A polygon is defined by a sequence of connected lines. A `MacPolygon` structure consists of two fixed-length fields followed by a variable-length array of points: the starting point followed by each successive point to which a line is drawn.

Your application typically does not need to create a `MacPolygon` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

PrinterFontStatus

```
struct PrinterFontStatus {
    SInt32 oResult;
    SInt16 iFondID;
    Style iStyle;
};
typedef struct PrinterFontStatus PrinterFontStatus;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

PrinterScalingStatus

```
struct PrinterScalingStatus {
    Point oScalingFactors;
};
typedef struct PrinterScalingStatus PrinterScalingStatus;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

PrinterStatusOpcode

```
typedef SInt32 PrinterStatusOpcode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDArcUPP

```
typedef QDArcProcPtr QDArcUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDBitsUPP

```
typedef QDBitsProcPtr QDBitsUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDByte

```
typedef SignedByte QDByte;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDCommentUPP

```
typedef QDCommentProcPtr QDCommentUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDErr

```
typedef short QDErr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDGetPicUPP

```
typedef QDGetPicProcPtr QDGetPicUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDGlobals

```
struct QDGlobals {
    char privates[76];
    long randSeed;
    BitMap screenBits;
    Cursor arrow;
    Pattern dkGray;
    Pattern ltGray;
    Pattern gray;
    Pattern black;
    Pattern white;
    GrafPtr thePort;
};
typedef struct QDGlobals QDGlobals;
typedef QDGlobals * QDGlobalsPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDJShieldCursorUPP

```
typedef QDJShieldCursorProcPtr QDJShieldCursorUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDLineUPP

```
typedef QDLineProcPtr QDLineUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDOpcodeUPP

```
typedef QDOpcodeProcPtr QDOpcodeUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDOvalUPP

```
typedef QDOvalProcPtr QDOvalUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDPictRef

Defines an opaque data type that represents a QuickDraw picture in the Quartz 2D graphics environment.

```
typedef struct QDPict * QDPictRef;
```

Discussion
This opaque type is used to draw QuickDraw picture data in a Quartz context. (Quartz 2D defines an analogous opaque type called `CGPDFDocumentRef` (page 340) which is used to draw PDF data in a Quartz context.) An instance of the `QDPictRef` type is called a **QDPict picture**. There are two ways to create a QDPict picture:

- You can call `QDPictCreateWithProvider` (page 2766), passing in a Quartz data provider for the picture data. Typically the source of this data is a `'PICT'` resource.
- You can call `QDPictCreateWithURL` (page 2767), passing in a Core Foundation URL that specifies a file with picture data in the data fork.

Both functions verify that picture header information is present, starting at either byte 1 or byte 513 of the picture data.

To draw a QDPict picture in a Quartz context, you call `QDPictDrawToCGContext` (page 2767). To get the bounds or native resolution of a QDPict picture, you call `QDPictGetBounds` (page 2768) or `QDPictGetResolution` (page 2769).

When you draw a QDPict picture in a PDF context, you can save the drawing in a PDF file. This is the recommended way to convert QuickDraw pictures into single-page PDF documents.

These additional sources of information may be helpful:

- The sample Carbon program *CGDrawPicture* shows how to use this opaque type to draw QuickDraw pictures in a Quartz context.
- For general information about QuickDraw pictures and the PICT graphics format, see Chapter 7 in *Inside Macintosh: Imaging With QuickDraw*.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QDPictToCGContext.h`

QDPolyUPP

```
typedef QDPolyProcPtr QDPolyUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDPrinterStatusUPP

```
typedef QDPrinterStatusProcPtr QDPrinterStatusUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDProcs

```
struct QDProcs {
    QDTextUPP textProc;
    QDLineUPP lineProc;
    QDRectUPP rectProc;
    QDRRectUPP rRectProc;
    QDOvalUPP ovalProc;
    QDArcUPP arcProc;
    QDPolyUPP polyProc;
    QDRgnUPP rgnProc;
    QDBitsUPP bitsProc;
    QDCommentUPP commentProc;
    QDTxMeasUPP txMeasProc;
    QDGetPicUPP getPicProc;
    QDPutPicUPP putPicProc;
};
typedef struct QDProcs QDProcs;
typedef QDProcs * QDProcsPtr;
```

Fields

`textProc`
A pointer to the low-level function that draws text. The standard QuickDraw function is the `StdText` function.

`lineProc`
A pointer to the low-level function that draws lines. The standard QuickDraw function is the `StdLine` (page 2818) function.

`rectProc`
A pointer to the low-level function that draws rectangles. The standard QuickDraw function is the `StdRect` (page 2821) function.

`rRectProc`
: A pointer to the low-level function that draws rounded rectangles. The standard QuickDraw function is the `StdRRect` (page 2822) function.

`ovalProc`
: A pointer to the low-level function that draws ovals. The standard QuickDraw function is the `StdOval` (page 2819) function.

`arcProc`
: A pointer to the low-level function that draws arcs. The standard QuickDraw function is the `StdArc` (page 2815) function.

`polyProc`
: A pointer to the low-level function that draws polygons. The standard QuickDraw function is the `StdPoly` (page 2820) function.

`rgnProc`
: A pointer to the low-level function that draws regions. The standard QuickDraw function is the `StdRgn` (page 2822) function.

`bitsProc`
: A pointer to the low-level function that copies bitmaps. The standard QuickDraw function is the `StdBits` (page 2816) function.

`commentProc`
: A pointer to the low-level function for processing a picture comment. The standard QuickDraw function is the `StdComment` (page 2817) function.

`txMeasProc`
: A pointer to the low-level function for measuring text width. The standard QuickDraw function is the `StdTxMeas` function.

`getPicProc`
: A pointer to the low-level function for retrieving information from the definition of a picture. The standard QuickDraw function is the `StdGetPic` (page 2817) function.

`putPicProc`
: A pointer to the low-level function for saving information as the definition of a picture. The standard QuickDraw function is the `StdPutPic` (page 2820) function.

Discussion
You need to use the `QDProcs` structure only if you customize one or more of QuickDraw's low-level drawing functions. Use `SetStdProcs` (page 2812) to create a `QDProcs` structure.

The `QDProcs` structure contains pointers to low-level drawing functions. You can change the fields of this structure to point to functions of your own devising.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QDPutPicUPP

```
typedef QDPutPicProcPtr QDPutPicUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDRectUPP

```
typedef QDRectProcPtr QDRectUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDRegionBitsRef

```
typedef struct OpaqueQDRegionBitsRef * QDRegionBitsRef;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
QuickdrawAPI.h

QDRegionParseDirection

```
typedef SInt32 QDRegionParseDirection;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawAPI.h

QDRgnUPP

```
typedef QDRgnProcPtr QDRgnUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDRRectUPP

```
typedef QDRRectProcPtr QDRRectUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDStdGlyphsUPP

```
typedef QDStdGlyphsProcPtr QDStdGlyphsUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDTextUPP

```
typedef QDTextProcPtr QDTextUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

QDTxMeasUPP

```
typedef QDTxMeasProcPtr QDTxMeasUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

RegionToRectsUPP

```
typedef RegionToRectsProcPtr RegionToRectsUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawAPI.h

ReqListRec

```
struct ReqListRec {
    short reqLSize;
    short reqLData[1];
};
typedef struct ReqListRec ReqListRec;
```

Fields

`reqLSize`

The size of this `ReqListRec` data structure minus one.

`reqLData`

An array of integers representing offsets into a color table.

Discussion

The `ReqListRec` data structure is a parameter to the `SaveEntries` function by which you can describe color table entries to be saved.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickdrawTypes.h`

RGBColor

```
struct RGBColor {
    unsigned short red;
    unsigned short green;
    unsigned short blue;
};
typedef struct RGBColor RGBColor;
typedef RGBColor * RGBColorPtr;
```

Fields

`red`

An unsigned integer specifying the red value of the color.

`green`

An unsigned integer specifying the green value of the color.

`blue`

An unsigned integer specifying the blue value of the color.

Discussion

You usually specify a color to QuickDraw by creating an `RGBColor` structure in which you assign the red, green, and blue values of the foreground color. For example, when you want to set the foreground color for drawing, you create an `RGBColor` structure that defines the foreground color you desire; then you pass that structure as a parameter to the `RGBForeColor` function.

In an `RGBColor` structure, three 16-bit unsigned integers give the intensity values for the three additive primary colors.

Availability

Available in Mac OS X v10.0 and later.

Declared In
IOMacOSTypes.h

RgnHandle

An opaque type that represents a QuickDraw region.

```
typedef struct OpaqueRgnHandle * RgnHandle;
```

Discussion
A region is an arbitrary area or set of areas on the QuickDraw coordinate plane. The outline of a region should be one or more closed loops.

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

SProcRec

```
struct SProcRec {
    Handle nxtSrch;
    ColorSearchUPP srchProc;
};
typedef struct SProcRec SProcRec;
typedef SProcRec * SProcPtr;
```

Fields
nxtSrch
A handle to the next SProcRec data structure in the chain of search functions.
srchProc
A pointer to a custom search function (described in ColorSearchProcPtr (page 2834)).

Discussion
The SProcRec data structure contains a pointer to a custom search function and a handle to the next SProcRec data structure in the function list.

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

WindowPtr

An opaque type that represents a window.

```
typedef struct OpaqueWindowPtr * WindowPtr;
```

Discussion
This is a Window Manager data type, defined in QuickDraw for historical reasons. Its role in Mac OS X is to serve as the basis for the widely used WindowRef data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

xColorSpec

```
struct xColorSpec {
    short value;
    RGBColor rgb;
    short xalpha;
};
typedef struct xColorSpec xColorSpec;
typedef xColorSpec * xColorSpecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

xCSpecArray

```
typedef xColorSpec xCSpecArray[1];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

Constants

chunky

```
enum {
    chunky = 0,
    chunkyPlanar = 1,
    planar = 2
};
```

Color Constants

```
enum {
    blackColor = 33,
    whiteColor = 30,
    redColor = 205,
    greenColor = 341,
    blueColor = 409,
    cyanColor = 273,
    magentaColor = 137,
    yellowColor = 69
};
```

Constants

`blackColor`
Represents black.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`whiteColor`
Represents white.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`redColor`
Represents red.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`greenColor`
Represents green.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`blueColor`
Represents blue.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`cyanColor`
Represents cyan.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`magentaColor`
Represents magenta.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`yellowColor`
Represents yellow.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

Discussion
These constants are used in the `color` parameter of the `ForeColor` (page 2628) and `BackColor` (page 2574) functions to specify one of the eight basic QuickDraw colors.

colorXorXFer

```
enum {
    colorXorXFer = 52,
    noiseXFer = 53,
    customXFer = 54
};
```

Cursor ID Constants

```
enum {
    sysPatListID = 0,
    iBeamCursor = 1,
    crossCursor = 2,
    plusCursor = 3,
    watchCursor = 4
};
```

Constants
`iBeamCursor`
The I-beam cursor; to select text
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`crossCursor`
The crosshairs cursor; to draw graphics
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`plusCursor`
The plus sign cursor; to select cells

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`watchCursor`
The wristwatch cursor; to indicate a short operation in progress

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

Discussion
When passing a value to the `Show_Cursor` function, use the `Cursors` data type to represent the kind of cursor to show.

cursorDoesAnimate

```
enum {
    cursorDoesAnimate = 1L << 0,
    cursorDoesHardware = 1L << 1,
    cursorDoesUnreadableScreenBits = 1L << 2
};
```

Device Attribute Constants

```
enum {
    interlacedDevice = 2,
    hwMirroredDevice = 4,
    roundedDevice = 5,
    hasAuxMenuBar = 6,
    burstDevice = 7,
    ext32Device = 8,
    ramInit = 10,
    mainScreen = 11,
    allInit = 12,
    screenDevice = 13,
    noDriver = 14,
    screenActive = 15,
    hiliteBit = 7,
    pHiliteBit = 0,
    defQDColors = 127,
    RGBDirect = 16,
    baseAddr32 = 4
};
```

Constants
`burstDevice`
If this bit is set to 1, the graphics device supports block transfer.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`ext32Device`
If this bit is set to 1, the graphics device must be used in 32-bit mode.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`ramInit`
If this bit is set to 1, the graphics device has been initialized from RAM.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`mainScreen`
If this bit is set to 1,the graphics device is the main screen.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`allInit`
If this bit is set to 1, all graphics devices were initialized from the `'scrn'` resource.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`screenDevice`
If this bit is set to 1, the graphics device is a screen.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`noDriver`
If this bit is set to 1, the `GDevice` structure has no driver.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`screenActive`
If this bit is set to 1, the graphics device is active.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

Discussion
These constants are used in the `attribute` parameters of the `SetDeviceAttribute` (page 2793) and `TestDeviceAttribute` (page 2826) functions, and in the `deviceFlags` parameter of the `DeviceLoopDrawingProcPtr` (page 2836) callback. These constants represent the `GDevice` structure's attributes, as bits in the `gdFlags` field.

Device Loop Flags

```
enum {
    singleDevices = 1 << singleDevicesBit,
    dontMatchSeeds = 1 << dontMatchSeedsBit,
    allDevices = 1 << allDevicesBit
};
```

Constants

`singleDevices`

If this flag is not set, `DeviceLoop` calls your drawing function only once for each set of similar graphics devices, and the first one found is passed as the target device. (It is assumed to be representative of all the similar graphics devices.) If you set the `singleDevices` flag, then `DeviceLoop` does not group similar graphics devices, (that is, those having identical pixel depths, black-and-white or color settings, and matching color table seeds), when it calls your drawing function.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`dontMatchSeeds`

If you set the `dontMatchSeeds` flag, then `DeviceLoop` does not consider the `ctSeed` field of `ColorTable` structures for graphics devices when comparing them; `DeviceLoop` ignores this flag if you set the `singleDevices` flag.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`allDevices`

If you set the `allDevices` flag, `DeviceLoop` ignores the `drawingRgn` parameter and calls your drawing function for every device. The value of the current graphics port's `visRgn` field is not affected when you set this flag.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

Discussion

When you use the `DeviceLoop` (page 2597) function, you can change its default behavior by using the `flags` parameter to specify one or more members of the set of flags defined by the `DeviceLoopFlags` data type. If you want to use the default behavior of `DeviceLoop`, specify 0 in the `flags` parameter.

deviceIsIndirect

```
enum {
    deviceIsIndirect = (1L << 0),
    deviceNeedsLock = (1L << 1),
    deviceIsStatic = (1L << 2),
    deviceIsExternalBuffer = (1L << 3),
    deviceIsDDSurface = (1L << 4),
    deviceIsDCISurface = (1L << 5),
    deviceIsGDISurface = (1L << 6),
    deviceIsAScreen = (1L << 7),
    deviceIsOverlaySurface = (1L << 8)
};
```

Drag Constraint Constants

When passed to the `DragControl` function, specify how a user can move a control.

```
enum {
    kNoConstraint = 0,
    kVerticalConstraint = 1,
    kHorizontalConstraint = 2
};
```

Constants

`kNoConstraint`
No constraint.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`kVerticalConstraint`
Constrain movement to horizontal axis only.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`kHorizontalConstraint`
Constrain movement to vertical axis only.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

Graphics Device Type Constants

```
enum {
    picLParen = 0,
    picRParen = 1,
    clutType = 0,
    fixedType = 1,
    directType = 2,
    gdDevType = 0
};
```

Constants

`clutType`
Represents a CLUT device--that is, one with colors mapped with a color lookup table.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`fixedType`
Represents a fixed colors device --that is, the color lookup table can't be changed.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`directType`
Represents a device with direct RGB colors.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`gdDevType`
If this bit is set to 0, the graphics device is black and white; if it is set to 1, the graphics device supports color.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

Discussion
These constants represent the general type of graphics device for the `gdType` field of the `GDevice` (page 2859) structure.

Graphics World Flags

Specify additional information passed to and from `NewGWorld` (page 2715) and related functions in parameters of type `GWorldFlags` (page 2863).

```
enum {
    pixPurge = 1L << pixPurgeBit,
    noNewDevice = 1L << noNewDeviceBit,
    useTempMem = 1L << useTempMemBit,
    keepLocal = 1L << keepLocalBit,
    useDistantHdwrMem = 1L << useDistantHdwrMemBit,
    useLocalHdwrMem = 1L << useLocalHdwrMemBit,
    pixelsPurgeable = 1L << pixelsPurgeableBit,
    pixelsLocked = 1L << pixelsLockedBit,
    kNativeEndianPixMap = 1L << nativeEndianPixMapBit,
    kAllocDirectDrawSurface = 1L << 14,
    mapPix = 1L << mapPixBit,
    newDepth = 1L << newDepthBit,
    alignPix = 1L << alignPixBit,
    newRowBytes = 1L << newRowBytesBit,
    reallocPix = 1L << reallocPixBit,
    clipPix = 1L << clipPixBit,
    stretchPix = 1L << stretchPixBit,
    ditherPix = 1L << ditherPixBit,
    gwFlagErr = 1L << gwFlagErrBit
};
```

Constants

`pixPurge`

If you specify this flag for the `flags` parameter of the `NewGWorld` function, `UpdateGWorld` (page 2831) makes the base address for the offscreen pixel image purgeable.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`noNewDevice`

If you specify this flag for the `flags` parameter of the `UpdateGWorld` (page 2831) function, `NewGWorld` does not create a new offscreen `GDevice` structure; instead, `NewGWorld` uses either the `GDevice` structure you specify or the `GDevice` structure for a video card on the user's system.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`useTempMem`

If you specify this in the `flags` parameter of the `UpdateGWorld` (page 2831) function, `NewGWorld` creates the base address for an offscreen pixel image in temporary memory. You generally should not use this flag. You should use temporary memory only for fleeting purposes and only with the `GetPixelsState` (page 2651) function so that other applications can launch.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`keepLocal`

If you specify this in the `flags` parameter of the `UpdateGWorld` (page 2831) function, `NewGWorld` creates a pixel image in Macintosh main memory where it cannot be cached to a graphics accelerator card.

If you specify this in the flags parameter of `GetPixelsState` (page 2651), `UpdateGWorld` keeps the offscreen pixel image in Macintosh main memory.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`pixelsPurgeable`
: If you specify this in the `state` parameter of the `UpdateGWorld` (page 2831) function, `SetPixelsState` makes the base address for an offscreen pixel map purgeable. If you use the `SetPixelsState` function without passing it this flag, then `SetPixelsState` makes the base address for an offscreen pixel map unpurgeable. If the `GetPixelsState` (page 2651) function returns this flag, then the base address for an offscreen pixel is purgeable.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`pixelsLocked`
: If you specify this flag for the `state` parameter of the `SetPixelsState` function, `SetPixelsState` locks the base address for an offscreen pixel image. If you use the `SetPixelsState` function without passing it this flag, then `SetPixelsState` unlocks the offscreen pixel image. If the `GetPixelsState` function returns this flag, then the base address for an offscreen pixel is locked.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`kNativeEndianPixMap`
: By default, the function `NewGWorld` (page 2715) allocates pixel buffers with big-endian byte ordering regardless of the system architecture. If this flag is passed in the `flags` parameter of `NewGWorld`, the pixel format will be set to `k32ARGBPixelFormat` or `k16BE555PixelFormat` on a PowerPC system, and to `k32BGRAPixelFormat` or `k16LE555PixelFormat` on an Intel system, for depths 32 or 16, respectively. Note that `NewGWorld` is the only function where this flag is observed; `NewGWorldFromPtr` (page 2718) and `UpdateGWorld` (page 2831) ignore it.

Available in Mac OS X v10.3 and later.

Declared in `QDOffscreen.h`.

`mapPix`
: If the `UpdateGWorld` (page 2831) function returns this flag, then it remapped the colors in the offscreen pixel map to a new color table.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`newDepth`
: If the `UpdateGWorld` function returns this flag, then it translated the offscreen pixel map to a different pixel depth.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`alignPix`
: If the `UpdateGWorld` function returns this flag, then it realigned the offscreen pixel image to an onscreen window.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`newRowBytes`
: If the `UpdateGWorld` function returns this flag, then it changed the `rowBytes` field of the `PixMap` structure for the offscreen graphics world.

Available in Mac OS X v10.0 and later.

Declared in `QDOffscreen.h`.

`reallocPix`
: If the `UpdateGWorld` function returns this flag, then it reallocated the base address for the offscreen pixel image. Your application should then reconstruct the pixel image or draw directly in a window instead of preparing the image in an offscreen graphics world.

  Available in Mac OS X v10.0 and later.

  Declared in `QDOffscreen.h`.

`clipPix`
: If you specify this flag in the `flags` parameter of the `UpdateGWorld` (page 2831) function, then `UpdateGWorld` updates and clips the pixel image to the new boundary rectangle specified. If the `UpdateGWorld` function returns this flag, then it clipped the pixel image.

  Available in Mac OS X v10.0 and later.

  Declared in `QDOffscreen.h`.

`stretchPix`
: If you specify this flag in the `flags` parameter of the `UpdateGWorld` (page 2831) function, then `UpdateGWorld` updates and stretches or shrinks the pixel image to the new boundary rectangle specified. If the `UpdateGWorld` function returns this flag, then it stretched or shrank the offscreen image.

  Available in Mac OS X v10.0 and later.

  Declared in `QDOffscreen.h`.

`ditherPix`
: If you specify this flag in the `flags` parameter of the `UpdateGWorld` (page 2831) function, then `UpdateGWorld` dithers the pixel image to the new boundary rectangle specified. Include this flag with the `clipPix` or `stretchPix` flag. If the `UpdateGWorld` function returns this flag, then it dithered the offscreen image.

  Available in Mac OS X v10.0 and later.

  Declared in `QDOffscreen.h`.

`gwFlagErr`
: If the `UpdateGWorld` function returns this flag, then it was unsuccessful and the offscreen graphics world was left unchanged.

  Available in Mac OS X v10.0 and later.

  Declared in `QDOffscreen.h`.

invalColReq

```
enum {
    invalColReq = -1
};
```

italicBit

```
enum {
    italicBit = 1,
    ulineBit = 2,
    outlineBit = 3,
    shadowBit = 4,
    condenseBit = 5,
    extendBit = 6
};
```

Pixel Formats

```
enum {
    k16LE555PixelFormat = 'L555',
    k16LE5551PixelFormat = '5551',
    k16BE565PixelFormat = 'B565',
    k16LE565PixelFormat = 'L565',
    k24BGRPixelFormat = '24BG',
    k32BGRAPixelFormat = 'BGRA',
    k32ABGRPixelFormat = 'ABGR',
    k32RGBAPixelFormat = 'RGBA',
    kYUVSPixelFormat = 'yuvs',
    kYUVUPixelFormat = 'yuvu',
    kYVU9PixelFormat = 'YVU9',
    kYUV411PixelFormat = 'Y411',
    kYVYU422PixelFormat = 'YVYU',
    kUYVY422PixelFormat = 'UYVY',
    kYUV211PixelFormat = 'Y211',
    k2vuyPixelFormat = '2vuy'
};
```

k1MonochromePixelFormat

```
enum {
    k1MonochromePixelFormat = 0x00000001,
    k2IndexedPixelFormat = 0x00000002,
    k4IndexedPixelFormat = 0x00000004,
    k8IndexedPixelFormat = 0x00000008,
    k16BE555PixelFormat = 0x00000010,
    k24RGBPixelFormat = 0x00000018,
    k32ARGBPixelFormat = 0x00000020,
    k1IndexedGrayPixelFormat = 0x00000021,
    k2IndexedGrayPixelFormat = 0x00000022,
    k4IndexedGrayPixelFormat = 0x00000024,
    k8IndexedGrayPixelFormat = 0x00000028
};
```

kCursorComponentInit

```
enum {
    kCursorComponentInit = 0x0001,
    kCursorComponentGetInfo = 0x0002,
    kCursorComponentSetOutputMode = 0x0003,
    kCursorComponentSetData = 0x0004,
    kCursorComponentReconfigure = 0x0005,
    kCursorComponentDraw = 0x0006,
    kCursorComponentErase = 0x0007,
    kCursorComponentMove = 0x0008,
    kCursorComponentAnimate = 0x0009,
    kCursorComponentLastReserved = 0x0050
};
```

kCursorComponentsVersion

```
enum {
    kCursorComponentsVersion = 0x00010001
};
```

kCursorComponentType

```
enum {
    kCursorComponentType = 'curs'
};
```

kCursorImageMajorVersion

```
enum {
    kCursorImageMajorVersion = 0x0001,
    kCursorImageMinorVersion = 0x0000
};
```

kPrinterFontStatus

```
enum {
    kPrinterFontStatus = 0,
    kPrinterScalingStatus = 1
};
```

kQDGrafVerbFrame

```
enum {
    kQDGrafVerbFrame = 0,
    kQDGrafVerbPaint = 1,
    kQDGrafVerbErase = 2,
    kQDGrafVerbInvert = 3,
    kQDGrafVerbFill = 4
};
```

kQDParseRegionFromTop

```
enum {
    kQDParseRegionFromTop = (1 << 0),
    kQDParseRegionFromBottom = (1 << 1),
    kQDParseRegionFromLeft = (1 << 2),
    kQDParseRegionFromRight = (1 << 3),
    kQDParseRegionFromTopLeft = kQDParseRegionFromTop | kQDParseRegionFromLeft,
    kQDParseRegionFromBottomRight = kQDParseRegionFromBottom |
kQDParseRegionFromRight
};
```

kQDRegionToRectsMsgInit

```
enum {
    kQDRegionToRectsMsgInit = 1,
    kQDRegionToRectsMsgParse = 2,
    kQDRegionToRectsMsgTerminate = 3
};
```

kQDUseDefaultTextRendering

```
enum {
    kQDUseDefaultTextRendering = 0,
    kQDUseTrueTypeScalerGlyphs = (1 << 0),
    kQDUseCGTextRendering = (1 << 1),
    kQDUseCGTextMetrics = (1 << 2),
    kQDSupportedFlags = kQDUseTrueTypeScalerGlyphs | kQDUseCGTextRendering  |
kQDUseCGTextMetrics,
    kQDDontChangeFlags = 0xFFFFFFFF
};
```

kRenderCursorInHardware

```
enum {
    kRenderCursorInHardware = 1L << 0,
    kRenderCursorInSoftware = 1L << 1
};
```

kXFer1PixelAtATime

```
enum {
    kXFer1PixelAtATime = 0x00000001,
    kXFerConvertPixelToRGB32 = 0x00000002
};
```

normalBit

```
enum {
    normalBit = 0,
```

```
    inverseBit = 1,
    redBit = 4,
    greenBit = 3,
    blueBit = 2,
    cyanBit = 8,
    magentaBit = 7,
    yellowBit = 6,
    blackBit = 5
};
```

pixPurgeBit

```
enum {
    pixPurgeBit = 0,
    noNewDeviceBit = 1,
    useTempMemBit = 2,
    keepLocalBit = 3,
    useDistantHdwrMemBit = 4,
    useLocalHdwrMemBit = 5,
    pixelsPurgeableBit = 6,
    pixelsLockedBit = 7,
    nativeEndianPixMapBit = 8,
    mapPixBit = 16,
    newDepthBit = 17,
    alignPixBit = 18,
    newRowBytesBit = 19,
    reallocPixBit = 20,
    clipPixBit = 28,
    stretchPixBit = 29,
    ditherPixBit = 30,
    gwFlagErrBit = 31
};
```

singleDevicesBit

```
enum {
    singleDevicesBit = 0,
    dontMatchSeedsBit = 1,
    allDevicesBit = 2
};
```

Source, Pattern, and Arithmetic Transfer Mode Constants

```
enum {
    srcCopy = 0,
    srcOr = 1,
    srcXor = 2,
    srcBic = 3,
    notSrcCopy = 4,
    notSrcOr = 5,
    notSrcXor = 6,
    notSrcBic = 7,
    patCopy = 8,
```

```
    patOr = 9,
    patXor = 10,
    patBic = 11,
    notPatCopy = 12,
    notPatOr = 13,
    notPatXor = 14,
    notPatBic = 15,
    grayishTextOr = 49,
    hilitetransfermode = 50,
    hilite = 50,
    blend = 32,
    addPin = 33,
    addOver = 34,
    subPin = 35,
    addMax = 37,
    adMax = 37,
    subOver = 38,
    adMin = 39,
    ditherCopy = 64,
    transparent = 36
};
```

Constants

`srcCopy`

For basic graphics ports, force the destination pixel black where the source pixel is black; where the source pixel is white, force the destination pixel white.

For color graphics ports, determines how close the color of the source pixel is to black, and assigns this relative amount of foreground color to the destination pixel. Determines how close the color of the source pixel is to white, and assigns this relative amount of background color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`srcOr`

For basic graphics ports, forces the destination pixel black if the source pixel is black; where the source pixel is white, leaves the destination pixel unaltered.

For color graphics ports, determines how close the color of the source pixel is to black, and assigns this relative amount of foreground color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`srcXor`

For basic and color graphics ports, inverts destination pixel where the source pixel is black. For a basic graphics port, where the source pixel is white, leaves the destination pixel unaltered.

For a color graphics port, for a colored destination pixel, uses the complement of its color if the pixel is direct, or inverts its index if the pixel is indexed.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`srcBic`

For a basic graphics port, forces destination pixel white where source pixel is black; where source pixel is white, leaves the destination pixel unaltered.

For a color graphics port, determines how close the color of the source pixel is to black, and assigns this relative amount of background color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notSrcCopy`

For a basic graphics port, forces the destination pixel white where the source pixel is black; where the source pixel is white, forces the destination pixel black.

For a color graphics port, determines how close the color of the source pixel is to black, and assigns this relative amount of background color to the destination pixel. Determines how close the color of the source pixel is to white, and assigns this relative amount of foreground color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notSrcOr`

For a basic graphics port, leaves the destination pixel unaltered where the source pixel is black; where the source pixel is white, forces the destination pixel black.

For a color graphics port, determines how close the color of the source pixel is to white, and assigns this relative amount of foreground color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notSrcXor`

For basic and color graphics ports, where the source pixel is white, inverts the destination pixel. For a basic graphics port, where the source pixel is black, leaves the destination pixel unaltered.

For a color graphics port, for a colored destination pixel, uses the complement of its color if the pixel is direct, or inverts its index if the pixel is indexed.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notSrcBic`

For a basic graphics port, where the source pixel is black, leaves the destination pixel unaltered; where the source pixel is white, forces the destination pixel white.

For a color graphics port, determines how close the color of the source pixel is to white, and assigns this relative amount of background color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`patCopy`

Where the pattern pixel is black, applies foreground color to the destination pixel; where the pattern pixel is white, applies background color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`patOr`

Where the pattern pixel is black, inverts the destination pixel; where the pattern pixel is white, leaves the destination pixel unaltered.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`patXor`

Where the pattern pixel is black, inverts the destination pixel; where the pattern pixel is white, leaves the destination pixel unaltered.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`patBic`

Where the pattern pixel is black, applies the background color to destination pixel; where the pattern pixel is white, leaves the destination pixel unaltered.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notPatCopy`

Where the pattern pixel is black, applies background color to destination pixel; where the pattern pixel is white, applies foreground color to the destination pixel

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notPatOr`

Where the pattern pixel is black, leaves the destination pixel unaltered; where the pattern pixel is white, applies foreground color to the destination pixel

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notPatXor`

Where the pattern pixel is black, leaves the destination pixel unaltered; where the pattern pixel is white, inverts the destination pixel

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`notPatBic`

Where the pattern pixel is black, leaves the destination pixel unaltered; where the pattern pixel is white, applies background color to the destination pixel.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`grayishTextOr`

Draws dimmed text on the screen. You can use it for black-and-white or color graphics ports. The `grayishTextOr` transfer mode is not considered a standard transfer mode because currently it is not stored in pictures, and printing with it is undefined. (It does not pass through the QuickDraw bottleneck functions.)

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`hilite`

Adds highlighting to the source or pattern mode. With highlighting, QuickDraw replaces the background color with the highlight color when your application copies images between graphics ports. This has the visual effect of using a highlighting pen to select the object. (The global variable `HiliteRGB` is read from parameter RAM when the machine starts. Basic graphics ports use the color stored in the `HiliteRGB` global variable as the highlight color. Color graphics ports default to the `HiliteRGB` global variable, but can be overridden by the `HiliteColor` function.)

For text, specifies that the caret position should be determined according to the primary line direction, based on the value of `SysDirection`.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`blend`

Replaces the destination pixel with a blend of the source and destination pixel colors. If the destination is a bitmap or 1-bit pixel map, reverts to `srcCopy` mode.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`addPin`

Replaces the destination pixel with the sum of the source and destination pixel colors-- up to a maximum allowable values. If the destination is a bitmap or 1-bit pixel map, reverts to `srcBic` mode.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`addOver`

Replaces the destination pixel with the sum of the source and destination pixel colors, except if the value of the red, green, or blue component exceeds 65,536, then `addOver` subtracts 65,536 from that value. If the destination is a bitmap or 1-bit pixel map, reverts to `srcXor` mode.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`subPin`

Replaces the destination pixel with the difference of the source and destination pixel colors, but not less than a minimum allowable value. If the destination is a bitmap or 1-bit pixel map, reverts to `srcOr` mode.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`addMax`

Compares the source and destination pixels, and replaces the destination pixel with the color containing the greater saturation of each of the RGB components. If the destination is a bitmap or 1-bit pixel map, reverts to `srcBic` mode.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`subOver`

Replaces the destination pixel with the difference of the source and destination pixel colors, except if the value of the red, green, or blue component is less than 0, then it adds the negative result to 65,536. if the destination is a bitmap or 1-bit pixel map, revert to `srcXor` mode.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`adMin`

Compares the source and destination pixels, and replaces the destination pixel with the color containing the lesser saturation of each of the RGB components. If the destination is a bitmap or 1-bit pixel map, reverts to `srcOr` mode.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`ditherCopy`

On computers running System 7, you can add dithering to any source mode by adding this constant or the value it represents to the source mode.

Dithering is a technique that mixes existing colors to create the effect of additional colors. It also improves images that you shrink or that you copy from a direct pixel device to an indexed device.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`transparent`

Replaces the destination pixel with the source pixel if the source pixel is not equal to the background color. The `transparent` mode replaces the destination pixel with the source pixel if the source pixel isn't equal to the background color. This mode is most useful in 8-bit, 4-bit, or 2-bit color modes.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

Discussion

`CopyBits` (page 2584) uses the source and arithmetic transfer mode constants in the `mode` parameter to specify the manner in which pixels are transferred from a source pixel map to a destination pixel map.

`PenMode` (page 2744) uses the pattern mode constants in the `mode` parameter to specify source modes for transferring the bits from a source bitmap to a destination bitmap.

The `TextMode` function uses these constants to set the transfer mode in the graphics port `txMode` field.

The transfer mode determines the interplay between what an application is drawing (the source) and what already exists on the display device (the destination), resulting in the text display.

There are two basic kinds of modes: pattern (`pat`) and source (`src`).

The pattern mode constants are `patCopy`, `patOr`, `patXor`, `patBic`, `notPatCopy`, `notPatOr`, `notPatXor`, and `notPatBic`.

Source is the kind that you use for drawing text. There are four basic Boolean operations: `Copy`, `Or`, `Xor`, and `Bic` (bit clear), each of which has an inverse variant in which the source is inverted before the transfer, yielding eight operations in all. Basic QuickDraw supports these eight transfer modes. Color QuickDraw interprets the source mode constants differently than basic QuickDraw does. Color QuickDraw enables your application to achieve color effects within those basic transfer modes, and offers an additional set of transfer modes that perform arithmetic operations on the RGB values of the source and destination pixels. Other transfer modes are `grayishTextOr`, `transparent` mode, and text mask mode.

The arithmetic transfer modes are `addOver`, `addPin`, `subOver`, `subPin`, `addMax`, `adMax`, `adMin`, and `blend`. For color, the arithmetic modes change the destination pixels by performing arithmetic operations on the source and destination pixels. Arithmetic transfer modes calculate pixel values by adding, subtracting, or averaging the RGB components of the source and destination pixels. They are most useful for 8-bit color, but they work on 4-bit and 2-bit color also. When the destination bitmap is one bit deep, the mode reverts to the basic transfer mode that best approximates the arithmetic mode requested.

Verb Constants

```
enum {
    frame = kQDGrafVerbFrame,
    paint = kQDGrafVerbPaint,
    erase = kQDGrafVerbErase,
    invert = kQDGrafVerbInvert,
    fill = kQDGrafVerbFill
};
```

Constants

`frame`
Specifies the frame action.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`paint`
Specifies the paint action.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`erase`
Specifies the erase action.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`invert`
Specifies the invert action.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

`fill`
Specifies the fill action.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawTypes.h`.

Discussion

When you use the `StdRect` (page 2821), `StdRRect` (page 2822), `StdOval` (page 2819), `StdArc` (page 2815), `StdPoly` (page 2820), or `StdRgn` (page 2822) functions, these constants are used in the `verb` parameter to specify the type of action taken by those low-level drawing functions.

Result Codes

The table below lists the result codes specific to QuickDraw.

Result Code	Value	Description
`updPixMemErr`	-125	Insufficient memory to update a pixmap. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
noMemForPictPlaybackErr	-145	Insufficient memory for drawing the picture. Available in Mac OS X v10.0 and later.
pixMapTooDeepErr	-148	Pixel map is deeper than 1 bit per pixel. Available in Mac OS X v10.0 and later.
nsStackErr	-149	Insufficient stack Available in Mac OS X v10.0 and later.
cMatchErr	-150	Color2Index failed to find an index. Available in Mac OS X v10.0 and later.
cTempMemErr	-151	Failed to allocate memory for temporary structures. Available in Mac OS X v10.0 and later.
cNoMemErr	-152	Failed to allocate memory for structures. Available in Mac OS X v10.0 and later.
cRangeErr	-153	Range error on color table requests. Available in Mac OS X v10.0 and later.
cProtectErr	-154	ColorTable structure entry protection violation. Available in Mac OS X v10.0 and later.
cDevErr	-155	Invalid type of graphics device. Available in Mac OS X v10.0 and later.
cResErr	-156	Invalid resolution for MakeITable. Available in Mac OS X v10.0 and later.
cDepthErr	-157	Invalid pixel depth. Available in Mac OS X v10.0 and later.
rgnTooBigErr	-500	Bitmap too large to convert to a region. Available in Mac OS X v10.0 and later.
kQDNoPalette	-3950	Available in Mac OS X v10.2 and later.
kQDNoColorHWCursorSupport	-3951	Available in Mac OS X v10.2 and later.
kQDCursorAlreadyRegistered	-3952	Available in Mac OS X v10.2 and later.
kQDCursorNotRegistered	-3953	Available in Mac OS X v10.2 and later.
kQDCorruptPICTDataErr	-3954	Available in Mac OS X v10.2 and later.

QuickDraw Text Reference (Not Recommended)

Framework: ApplicationServices/ApplicationServices.h

Declared in QuickdrawText.h

Important: The information in this document is obsolete and should not be used for new development.

Overview

You can use the QuickDraw text routines to measure and draw text ranging in complexity from a single glyph to a line of justified text containing multiple languages and styles. In addition to measuring and drawing text, the QuickDraw text routines also help you to determine which characters to highlight and where to position the caret to mark the insertion point. These routines translate pixel locations into byte offsets and vice versa.

To understand the routines described in this document, it is helpful to be familiar with the other parts of QuickDraw. It is also helpful to be familiar with the Font Manager, because of the close relationship between QuickDraw and the Font Manager. To understand the tasks involved in text layout, you should also be acquainted with the Text Utilities and the Script Manager.

Carbon supports the majority of QuickDraw Text.

Functions by Task

Determining the Caret Position, and Selecting and Highlighting Text

`CharToPixel` (page 2911) Deprecated in Mac OS X v10.4

Returns the screen pixel width from the left edge of a text segment to the glyph of the character whose byte offset you specify. (Deprecated. Use ATSUI instead.)

`HiliteText` (page 2921) Deprecated in Mac OS X v10.4

Finds all the characters between two byte offsets in a text segment whose glyphs are to be highlighted. (Deprecated. Use ATSUI instead.)

`PixelToChar` (page 2927) Deprecated in Mac OS X v10.4

Returns the byte offset of a character in a style run, or part of a style run, whose onscreen glyph is nearest the place where the user clicked the mouse. (Deprecated. Use ATSUI instead.)

Drawing Text

`DrawChar` (page 2914) Deprecated in Mac OS X v10.4
Draws the glyph for a single 1-byte character at the current pen location in the current graphics port. (Deprecated. Use ATSUI or Quartz instead.)

`DrawJustified` (page 2915) Deprecated in Mac OS X v10.4
Draws the specified text at the current pen location in the current graphics port, taking into account the adjustment necessary to condense or extend the text by the slop value, appropriately for the script system. (Deprecated. Use ATSUI instead.)

`DrawString` (page 2917) Deprecated in Mac OS X v10.4
Draws the text of the specified Pascal string at the pen location in the current graphics port (`GrafPort` or `CGrafPort`). (Deprecated. Use ATSUI or Quartz instead.)

`DrawText` (page 2918) Deprecated in Mac OS X v10.4
Draws the specified text at the current pen location in the current graphics port. (Deprecated. Use ATSUI or Quartz instead.)

`StandardGlyphs` (page 2933) Deprecated in Mac OS X v10.4
This obsolete function doesn't do anything in Mac OS X. (Deprecated. Use ATSUI to render Unicode text.)

`StdText` (page 2933) Deprecated in Mac OS X v10.4
Draws text from an arbitrary structure in memory. (Deprecated. Use ATSUI or Quartz instead.)

`stdtext` (page 2934) Deprecated in Mac OS X v10.4
Draws text from an arbitrary structure in memory. (Deprecated. Use ATSUI or Quartz instead.)

Laying Out a Line of Text

`GetFormatOrder` (page 2920) Deprecated in Mac OS X v10.4
Determines the display order of style runs for a line of text containing multiple style runs with mixed directions. (Deprecated. Use ATSUI instead.)

`PortionLine` (page 2930) Deprecated in Mac OS X v10.4
Determines the correct proportion of extra space to apply to the specified style run in a line of justified text; that is, how to distribute the total slop value for a line among the style runs on that line. (Deprecated. Use ATSUI instead.)

`VisibleLength` (page 2944) Deprecated in Mac OS X v10.4
Calculates the length, in bytes, of a given text segment, excluding trailing white space. (Deprecated. Use ATSUI instead.)

Measuring Text

`CharWidth` (page 2913) Deprecated in Mac OS X v10.4
Returns the width (horizontel extension), in pixels, of the specified character. (Deprecated. Use ATSUI instead.)

`MeasureJustified` (page 2923) Deprecated in Mac OS X v10.4
Calculates, for text that is expanded, condensed, or scaled, the onscreen width in pixels from the left edge of the text segment to the glyph of the character. (Deprecated. Use ATSUI instead.)

`MeasureText` (page 2925) Deprecated in Mac OS X v10.4
Calculates the width of the character's glyph in pixels from the left edge of the text segment. (Deprecated. Use ATSUI instead.)

`StdTxMeas` (page 2934) Deprecated in Mac OS X v10.4
Measures the width of scaled or unscaled text. (Deprecated. Use ATSUI instead.)

`StringWidth` (page 2936) Deprecated in Mac OS X v10.4
Returns the length, in pixels, of the specified text string. (Deprecated. Use ATSUI instead.)

`TextWidth` (page 2941) Deprecated in Mac OS X v10.4
Returns the length, in pixels, of the specified text. (Deprecated. Use ATSUI instead.)

Setting Text Characteristics

`CharExtra` (page 2910) Deprecated in Mac OS X v10.4
Specifies, for a color graphics port (`CGrafPort`), the number of pixels by which to widen (or narrow) the glyphs of each nonspace character in a style run. (Deprecated. Use ATSUI instead.)

`GetFontInfo` (page 2919) Deprecated in Mac OS X v10.4
Returns information about the current graphics port's font, taking into account the style and size in which the glyphs are to be drawn. (Deprecated. Use ATSUI instead.)

`SpaceExtra` (page 2932) Deprecated in Mac OS X v10.4
Specifies the number of pixels by which to widen (or narrow) each space in a style run to be drawn in the current graphics port. (Deprecated. Use ATSUI instead.)

`TextFace` (page 2939) Deprecated in Mac OS X v10.4
Sets the style of the font in which the text is to be drawn in the current graphics port. (Deprecated. Use ATSUI or Quartz instead.)

`TextFont` (page 2939) Deprecated in Mac OS X v10.4
Sets the font of the current graphics port in which the text is to be rendered. (Deprecated. Use ATSUI or Quartz instead.)

`TextMode` (page 2940) Deprecated in Mac OS X v10.4
Sets the transfer mode for drawing text in the current graphics port. (Deprecated. Use ATSUI or Quartz instead.)

`TextSize` (page 2941) Deprecated in Mac OS X v10.4
Sets the font size for text drawn in the current graphics port to the specified number of points. (Deprecated. Use ATSUI or Quartz instead.)

Truncating Strings and Breaking Lines

`StyledLineBreak` (page 2937) Deprecated in Mac OS X v10.4
Returns the proper location to break a line of text, taking into account script and language considerations, making use of tables in the string-manipulation (`'itl2'`) resource in its computations. (Deprecated. Use ATSUI instead.)

`TruncString` (page 2943) Deprecated in Mac OS X v10.4
Ensures that a Pascal string fits into the specified pixel width, by truncating the string as necessary. This function makes use of the current script and font. (Deprecated. Use CFString instead.)

`TruncText` (page 2943) Deprecated in Mac OS X v10.4
: Ensures that a text string fits into the specified pixel width, by truncating the string as necessary. This function makes use of the current script and font. (Deprecated. Use CFString instead.)

Working With Universal Procedure Pointers

`DisposeStyleRunDirectionUPP` (page 2914) Deprecated in Mac OS X v10.4
: Disposes of a universal procedure pointer (UPP) to a style run direction callback. (Deprecated. Use ATSUI to handle style runs.)

`InvokeStyleRunDirectionUPP` (page 2922) Deprecated in Mac OS X v10.4
: Calls your style run direction callback. (Deprecated. Use ATSUI to handle style runs.)

`NewStyleRunDirectionUPP` (page 2927) Deprecated in Mac OS X v10.4
: Creates a new universal procedure pointer (UPP) to a style run direction callback. (Deprecated. Use ATSUI to handle style runs.)

Functions

CharExtra

Specifies, for a color graphics port (`CGrafPort`), the number of pixels by which to widen (or narrow) the glyphs of each nonspace character in a style run. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void CharExtra (
   Fixed extra
);
```

Parameters

extra
: The amount (in pixels or decimal fractions of a pixel) to widen (or narrow) each glyph other than the space character in a range of text.

Discussion

The `CharExtra` function sets the value of the `chExtra` field of the color graphics port structure. This field contains a number that is in 4.12 fractional notation: four bits of signed integer followed by 12 bits of fraction. The `CharExtra` function uses the value of the `txSize` field, so you must call `TextSize` to set the font size of the text before you call `CharExtra`.

The initial setting is 0. You can pass a negative value for the `extra` parameter, but be careful not to narrow glyphs so much that the text is unreadable. The measuring and drawing functions use the value in this field when an application calls them to measure or draw text. The `CharExtra` function is available only for color graphic ports.

Do not use `CharExtra` for script systems that include zero-width characters, such as diacritical marks, because intercharacter space is added to all glyphs, separating the diacritical mark from the glyph of the character. Do not use it for script systems that include contextual forms, such as ligatures or conjunct characters, which would not be represented properly were intercharacter space added to these glyphs. For example, you should not use `CharExtra` for the Devanagari or Arabic languages, whose text is drawn as connected glyphs, or with the Sonata font because it includes zero-width characters.

The 2-byte script systems use the `chExtra` field value properly.

To ensure future compatibility and benefit from any enhancements, always use this function to modify the `chExtra` field, rather than directly change the field value.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

CharToPixel

Returns the screen pixel width from the left edge of a text segment to the glyph of the character whose byte offset you specify. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
short CharToPixel (
   Ptr textBuf,
   long textLength,
   Fixed slop,
   long offset,
   short direction,
   JustStyleCode styleRunPosition,
   Point numer,
   Point denom
);
```

Parameters

`textBuf`

A pointer to the beginning of the text segment.

`textLength`

The length in bytes of the entire text segment pointed to by `textBuf`. The `CharToPixel` function requires the context of the complete text in order to determine the correct value.

`slop`

The amount of slop for the text to be drawn. A positive value extends the text segment; a negative value condenses the text segment.

The value of this parameter is the number of pixels by which the width of the text segment is to be changed, after the text has been scaled. The `slop` is a signed value that specifies how smuch the text is to be extended or condensed. The `slop` is derived from the calculations made using the proportion returned from the `PortionLine` function for a style run. To measure or draw text that is not to be extended or condensed, pass a `slop` value of 0.

`offset`

The offset from `textBuf` to the character within the text segment whose display pixel location is to be measured. For 2-byte script systems, if the character whose position is to be measured is 2 bytes long, this is the offset of the first byte.

direction

This parameter specifies whether `CharToPixel` is to return the caret position for a character with a direction of left-to-right or right-to-left. A direction value of `hilite` indicates that `CharToPixel` is to use the caret position for the character direction that matches the primary line direction as specified by the `SysDirection` global variable.

styleRunPosition

The position on the line of this style run. The style run can be the only one on the line, the leftmost on the line, the rightmost on the line, or one between two other style runs.

This parameter specifies the position of the style run on the display line. It is used to determine the proportion of total slop to apply to a style run, measure or draw a line of justified text, identify where to break a line of text, and determine the caret position to mark an insertion point or highlight text.

The style run position parameter is meaningful only for those script systems that use intercharacter spacing for justification. For all other script systems, the parameter exists for future extensibility. Although the style run position parameter is not used, for example, for justifying text in the Roman script system, to allow for future compatibility, you should always specify the appropriate value for it for all calls that take it.

For those script systems that do use intercharacter spacing, space between style runs may be allocated differently depending upon whether the style run is leftmost, rightmost, or between two other style runs. For example, depending on the script system, if a style run occurs at the beginning or end of a line, extra space may not be added to the outer edge of the outermost glyph, whereas if a style run is interior to a line, all of the glyphs of the text may be treated the same: extra space is allocated to both sides of every glyph including those at either end of the style run.

The current implementations of simple script systems such as Roman and Cyrillic do not justify a line of text by changing the width of nonspace characters. Instead, they rely solely on the use of space characters: the same amount of extra width is added to (or subtracted from) every space whether the space is at the beginning or end of the line or interior to it.

See "Style Run Position Constants" (page 2951) for a list of the constants you can supply.

numer

A point giving the numerator for the horizontal and vertical scaling factors.

Both `numer` and `denom` are point values: `numer` specifies the numerator for the horizontal and vertical scaling factors, and `denom` specifies the denominator for the horizontal and vertical scaling factors. Together, these values specify the scaling factors for the text: `numer.v` over `denom.v` gives the vertical scaling (height), and `numer.h` over `denom.h` gives the horizontal scaling factors (width). You need to specify values for `numer` and `denom` even if you are not scaling the text. For unscaled text, you can specify scaling factors of 1, 1.

denom

A point giving the denominator for the horizontal and vertical scaling factors.

Return Value

The screen pixel width from the left edge of a text segment to the glyph of the character whose byte offset you specify.

Discussion

You use `CharToPixel` to find the onscreen pixel location at which to draw a caret and to identify the selection points for highlighting. The `CharToPixel` function returns the horizontal distance in pixels from the start of the range of text beginning with the byte offset at `textBuf` to the glyph corresponding to the character whose byte offset is specified by the `offset` parameter. The pixel location is relative to the beginning of the text segment, not the left margin of the display line. To get the actual display line pixel location of the glyph relative to the left margin, you add the pixel value that `CharToPixel` returns to the pixel location at

the end of the previous style run (on the left) in display order. In other words, you need to know the length of the text in pixels on the display line up to the beginning of the range of text that you call `CharToPixel` for, and then you add in the screen pixel width that `CharToPixel` returns.

You specify a value for `textLen` that is equal to the entire visible part of the style run on a line and includes trailing spaces if and only if they are displayed. They may not be displayed, for example, for the last style run in memory order, which is part of the line. Do not confuse the `textLen` parameter with the `offset`, which is the byte offset of the character within the text segment whose pixel location `CharToPixel` is to return.

If you use `CharToPixel` to get a caret position to mark the insertion point, you specify a value of `leftCaret` or `rightCaret` for the `direction` parameter. You can use the value of the `PixelToChar leadingEdge` flag to determine the `direction` parameter value.

If the `leadingEdge` flag is `FALSE`, you base the value of the `direction` parameter on the direction of the character at the byte offset in memory that precedes the one that `PixelToChar` returns; if `leadingEdge` is `TRUE`, you base the value of the `direction` parameter on the direction of the character at the byte offset that `PixelToChar` returns. If there isn't a character at the byte offset, you base the value of the `direction` parameter on the primary line direction as determined by the `SysDirection` global variable.

Be sure to pass the same values for `styleRunPosition` and the scaling factors (`numer` and `denom`) to `CharToPixel` that you pass to any of the other justification functions for this style run.

The `CharToPixel` function works with text in all script systems. For 1-byte contextual script systems, `CharToPixel` calculates the widths of any ligatures, reversals, and compound characters that need to be drawn.

Note that `textLen` is the number of bytes to be drawn, not the number of characters. Because 2-byte script systems also include characters consisting of only one byte, do not simply multiply the number of characters by 2 to determine this value; you must determine and specify the correct number of bytes.

Special Considerations

The `CharToPixel` function may move memory; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

CharWidth

Returns the width (horizontel extension), in pixels, of the specified character. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
short CharWidth (
   CharParameter ch
);
```

Parameters

ch

The character whose width is to be measured.

Return Value
The width (horizontel extension), in pixels, of the specified character.

Discussion
The `CharWidth` function includes the effects of the stylistic variations for the text set in the current graphics port. If you change any of these attributes after determining the glyph width but before actually drawing it, the predetermined width may not be correct. For a space character, `CharWidth` also includes the effect of `SpaceExtra`. For a nonspace character, `CharWidth` includes the effect of `CharExtra`.

Because it takes a single-byte value as the `ch` parameter, `CharWidth` works only for 1-byte simple script systems.

A series of calls to `CharWidth` in a contextual 1-byte font may give incorrect results, because the width of a text segment may be different from the sum of its individual character widths. In that case, to measure a line of text you should call `TextWidth`.

Do not use the `CharWidth` function for 2-byte script systems. If you want to measure the width of a single glyph in a 2-byte font, you should use `TextWidth`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

DisposeStyleRunDirectionUPP

Disposes of a universal procedure pointer (UPP) to a style run direction callback. (Deprecated in Mac OS X v10.4. Use ATSUI to handle style runs.)

```
void DisposeStyleRunDirectionUPP (
   StyleRunDirectionUPP userUPP
);
```

Parameters
userUPP

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawText.h`

DrawChar

Draws the glyph for a single 1-byte character at the current pen location in the current graphics port. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void DrawChar (
   CharParameter ch
);
```

Parameters

ch

The character code whose glyph is to be drawn.

Discussion

The `DrawChar` function draws a single character's glyph and then advances the pen by the width of the glyph. If the glyph isn't in the font, the font's missing symbol is drawn.

If you're drawing more than one character, it's faster to make one `DrawString` or `DrawText` call rather than a series of `DrawChar` calls.

Because it takes a single-byte value as the `ch` parameter, `DrawChar` works only for 1-byte script systems. If you want to draw the glyph of a single character in a 2-byte script, call either `DrawText`, `DrawString`, or `DrawJustified`.

A series of calls to `DrawChar` in a 1-byte complex script system can give incorrect results because a text string is not always a simple concatenation of a series of characters. In a contextual script, two different glyphs may be used to represent a single character in its contextual form and alone. To draw a sequence of text in a 1-byte complex script system, use `DrawText`, `DrawString`, or `DrawJustified` instead.

For 1-byte complex scripts, you can use `DrawChar` for special purposes, such as to include the isolated glyph of a character in a book's index, for example, to show a single glyph as it exists apart from contextual transformations.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

DrawJustified

Draws the specified text at the current pen location in the current graphics port, taking into account the adjustment necessary to condense or extend the text by the slop value, appropriately for the script system. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void DrawJustified (
   Ptr textPtr,
   long textLength,
   Fixed slop,
   JustStyleCode styleRunPosition,
   Point numer,
   Point denom
);
```

Parameters

textPtr

A pointer to the memory location of the beginning of the text to be drawn.

textLength

The number of bytes of text to be drawn.

Note that `textLength` is the number of bytes to be drawn, not the number of characters. Because 2-byte script systems also include characters consisting of only 1 byte, do not simply multiply the number of characters by 2 to determine this value; you must determine and specify the correct number of bytes.

slop

The amount of slop for the text to be drawn. A positive value extends the text segment; a negative value condenses the text segment. Pass the value assessed for this style run based on the proportion returned for it from `PortionLine`.

The value of this parameter is the number of pixels by which the width of the text segment is to be changed, after the text has been scaled. The `slop` is a signed value that specifies how much the text is to be extended or condensed. The `slop` is derived from the calculations made using the proportion returned from the `PortionLine` function for a style run. To measure or draw text that is not to be extended or condensed, pass a `slop` value of 0.

styleRunPosition

The position on the line of this style run. The style run can be the only one on the line, the leftmost on the line, the rightmost on the line, or one between two other style runs. Be sure to pass the same value that you pass to `PortionLine`.

This parameter specifies the position of the style run on the display line. It is used to determine the proportion of total slop to apply to a style run, measure or draw a line of justified text, identify where to break a line of text, and determine the caret position to mark an insertion point or highlight text.

The style run position parameter is meaningful only for those script systems that use intercharacter spacing for justification. For all other script systems, the parameter exists for future extensibility. Although the style run position parameter is not used, for example, for justifying text in the Roman script system, to allow for future compatibility, you should always specify the appropriate value for it for all calls that take it.

For those script systems that do use intercharacter spacing, space between style runs may be allocated differently depending upon whether the style run is leftmost, rightmost, or between two other style runs. For example, depending on the script system, if a style run occurs at the beginning or end of a line, extra space may not be added to the outer edge of the outermost glyph, whereas if a style run is interior to a line, all of the glyphs of the text may be treated the same: extra space is allocated to both sides of every glyph including those at either end of the style run.

The current implementations of simple script systems such as Roman and Cyrillic do not justify a line of text by changing the width of nonspace characters. Instead, they rely solely on the use of space characters: the same amount of extra width is added to (or subtracted from) every space whether the space is at the beginning or end of the line or interior to it.

See "Style Run Position Constants" (page 2951) for a list of the constants you can supply.

numer

A point giving the numerator for the horizontal and vertical scaling factors.

Both numer and denom are point values: numer specifies the numerator for the horizontal and vertical scaling factors, and denom specifies the denominator for the horizontal and vertical scaling factors. Together, these values specify the scaling factors for the text: numer.v over denom.v gives the vertical scaling (height), and numer.h over denom.h gives the horizontal scaling factors (width). You need to specify values for numer and denom even if you are not scaling the text. For unscaled text, you can specify scaling factors of 1, 1. Be sure to pass the same value that you pass to `PortionLine`.

denom

A point giving the denominator for the horizontal and vertical scaling factors. Be sure to pass the same value that you pass to `PortionLine`.

Discussion
The `DrawJustified` function is similar to the `DrawText` function, except that you use it to draw text that is expanded or condensed by the number of pixels specified by `slop`. The `DrawJustified` function is most commonly used to draw a line of justified text.

The `DrawJustified` function draws the specified text in the font, size, and style of the current graphics port, taking into account any scaling factors, and it distributes the slop appropriately for the script system. Regardless of the line direction of the text to be drawn, you place the pen at the left edge of the line before calling `DrawJustified` for the first style run. For all subsequent style runs on that line, QuickDraw advances the pen appropriately.

If `DrawJustified` changes the width of spaces, it temporarily resets the space extra (`spExtra`) value. It adds to the current value of the field, if any, the amount of extra space to be applied to each space character within the range of text in order to justify the text, based on calculations that take into account the slop value and all of the text characteristics. On exit, `DrawJustified` restores the original value.

The `DrawJustifie` d function works with text in all script systems. For example, to depict justified Arabic text, `DrawJustified` uses extension bars to create the additional width that is distributed as slop within a style run.

For 1-byte complex script systems, `DrawJustified` substitutes the proper ligatures, reversals, and compound characters as needed.

For 2-byte script systems that do not use space characters to delimit words, `DrawJustified` distributes the slop value in a manner appropriate to the script system. For script systems, such as Japanese, that use ideographic characters, `DrawJustified` distributes the additional screen pixel width appropriately for the text representation.

Special Considerations
The `DrawJustified` function may move memory; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

DrawString

Draws the text of the specified Pascal string at the pen location in the current graphics port (`GrafPort` or `CGrafPort`). (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void DrawString (
   ConstStr255Param s
);
```

Parameters
s
A Pascal string consisting of the text to be drawn.

Discussion
The `DrawString` function draws the string with its left edge at the current pen location, extending right. The final position of the pen location, after the text is drawn, is to the right of the rightmost glyph in the string. QuickDraw does not do any formatting, such as handling of carriage returns or line feeds.

Note that you can use `DrawString` only for a Pascal string containing a single style run.

QuickDraw temporarily stores on the stack all of the text you ask it to draw, even if the text is to be clipped. When drawing large font sizes or complex style variations, draw only what is visible on the screen. You can determine the number of characters whose corresponding glyphs actually fit on the screen by calling the `StringWidth` function to determine the length of the string before calling `DrawString`.

If you specify values in the graphics port `spExtra` or `chExtra` fields to change the width of space or nonspace characters, `DrawString` takes these values into account.

For right-to-left text, such as Hebrew or Arabic, QuickDraw draws the final (leftmost) glyph first, then moves to the right through all the glyphs, drawing the initial (rightmost) glyph last.

Note that you should not change the width of nonspace characters for 1-byte simple script systems with zero-width characters or 1-byte complex script systems. For more information, see `CharExtra` (page 2910).

For contextual script systems, `DrawString` substitutes the proper ligatures, reversals, and compound characters as needed. Inside a picture definition, `DrawString` can't have a `byteCount` greater than 255.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar
Simple DrawSprocket

Declared In
`QuickdrawText.h`

DrawText

Draws the specified text at the current pen location in the current graphics port. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void DrawText (
    const void * textBuf,
    short firstByte,
    short byteCount
);
```

Parameters

`textBuf`
A pointer to a buffer containing the text to be drawn.

`firstByte`
An offset from the start of the text buffer (`textBuf`) to the first byte of the text to be drawn.

byteCount

The number of bytes of text to be drawn. Inside a picture definition, `DrawText` cannot have a `byteCount` greater than 255.

For 2-byte script systems, note that `byteCount` is the number of bytes to be drawn, not the number of glyphs. Because 2-byte script systems also include characters consisting of only 1 byte, do not simply multiply the number of characters by 2 to determine this value; you must determine and specify the correct number of bytes.

Discussion

The `DrawText` function draws the text with the leftmost glyph at the current pen location, extending right. After QuickDraw draws the text, it sets the pen location to the right of the rightmost glyph.

QuickDraw temporarily stores on the stack all of the text you ask it to draw, even if the text is to be clipped. When drawing a range of text, it's best to draw only what is visible on the screen. If an entire text string does not fit on a line, truncate the text at a word boundary. If possible, avoid truncating within a style run. You can determine the number of characters whose glyphs actually fit on the screen by calling the `TextWidth` function before calling `DrawText`.

If you specify values in the graphics port `spExtra` and `chExtra` fields to change the width of nonspace and space characters, both `TextWidth` and `DrawText` take these values into account.

For 1-byte complex script systems, `DrawText` substitutes the proper ligatures, reversals, and compound characters as needed.

For right-to-left text, such as Hebrew or Arabic, QuickDraw draws the final (leftmost) glyph first, then moves to the right through all the characters, drawing the initial (rightmost) glyph last.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

GetFontInfo

Returns information about the current graphics port's font, taking into account the style and size in which the glyphs are to be drawn. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void GetFontInfo (
   FontInfo *info
);
```

Parameters

info

Pointer to a font information structure that contains the font measurement information, in integer values.

Discussion

The `GetFontInfo` function returns the ascent, descent, leading, and width of the largest glyph of the font in the text font, size, and style specified in the current graphics port. If the script system specified by the current graphics port `txFont` field has an associated font, as do Hebrew and Arabic, `GetFontInfo` returns combined information based on both fonts. This is to accommodate text written in the Roman script when

the primary script system is non-Roman. However, even if all of the text is written in a non-Roman script, if there is an associated font, `GetFontInfo` always bases its information on the combined fonts. You can determine the line height, in pixels, by adding the values of the ascent, descent, and leading fields.

The `GetFontInfo` function is similar to the Font Manager's `FontMetrics` function, except that the `GetFontInfo` function returns integer values. See `FontInfo` (page 2947) for a description of the structure and its fields.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

GetFormatOrder

Determines the display order of style runs for a line of text containing multiple style runs with mixed directions. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void GetFormatOrder (
   FormatOrderPtr ordering,
   short firstFormat,
   short lastFormat,
   Boolean lineRight,
   StyleRunDirectionUPP rlDirProc,
   Ptr dirParam
);
```

Parameters

ordering
A pointer to a format order array, with (`lastFormat - firstFormat + 1`) entries. The function fills the array with the display order of each style run. On exit, the array contains a permuted list of the numbers from `firstFormat` to `lastFormat`.

The first entry in the array is the number of the style run to draw first; this is the leftmost style run in display order. The last entry in the array is the number of the entry to draw last, the rightmost style run in display order.

Upon completion of the call, the `FormatOrder` (page 2948) array contains the numbers identifying the style runs in display order.

firstFormat
A number greater than or equal to 0 identifying the first style run in storage order that is part of the line for which you are calling `GetFormatOrder`.

lastFormat
A number greater than or equal to 0 identifying the last style run in storage order that is part of the line for which you are calling `GetFormatOrder`.

lineRight
A flag that you set to `TRUE` if the primary line direction is right-to-left.

rlDirProc

A pointer to a callback function that calculates the correct direction, given the style run identifier. The `GetFormatOrder` function calls the application-defined `rlDirProc` function for each identifier from `firstFormat` to `lastFormat`.

This function returns `TRUE` for right-to-left text direction and `FALSE` for left-to-right. Given `dirParam` and a style run identifier, the callback function should be able to determine the style run direction. For more information, see `StyleRunDirectionProcPtr` (page 2946).

dirParam

A pointer to a parameter block that contains the font and script information for each style run in the text. This parameter block is used by the application-supplied function.

Discussion

The `GetFormatOrder` function helps you determine how to draw text that contains multiple style runs with mixed directions. For mixed-directional text, after you determine where to break the line, you need to call `GetFormatOrder` to determine the display order. When you call `GetFormatOrder`, you supply a Boolean function, and reference it using the `rlDirProc` parameter. This function calculates the direction of each style run identified by number. Do not call `GetFormatOrder` if there is only one style run on the line.

You must index the style runs in storage order. You pass `GetFormatOrder` numbers identifying the first and last style runs of the line in storage order and the primary line direction. The `GetFormatOrder` function returns to you an equivalent sequence in display order.

If you do not explicitly define the primary line direction of the text, base the `lineRight` parameter on the value of the `SysDirection` global variable. (The `SysDirection` global variable is set to `-1` if the system direction is right to left, and 0 if the system direction is left to right.)

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

HiliteText

Finds all the characters between two byte offsets in a text segment whose glyphs are to be highlighted. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void HiliteText (
   Ptr textPtr,
   short textLength,
   short firstOffset,
   short secondOffset,
   OffsetTable offsets
);
```

Parameters

textPtr

A pointer to a buffer that contains the text to be highlighted.

textLength

The length in bytes of the entire text segment pointed to by `textPtr`.

firstOffset
The byte offset from `textPtr` to the first character to be highlighted.

secondOffset
The byte offset from `textPtr` to the last character to be highlighted.

offsets
A table that, upon completion of the call, specifies the boundaries of the text to be highlighted.

Discussion
The `HiliteText` function returns three pairs of byte offsets that mark the onscreen ranges of text to be highlighted. This is because for bidirectional text, although the characters are contiguous in memory, their displayed glyphs can include up to three separate ranges of text.

The `HiliteText` function takes into account the fact that to highlight the complete range of text whose beginning and ending byte offsets you pass it, it must return byte offsets that encompass the glyphs of the first and last characters in the text segment. To determine the correct offset pairs, `HiliteText` relies on the primary line direction as specified by the `SysDirection` global variable.

Before calling `HiliteText`, you must set up an offset table (of type `OffsetTable`) in your application to hold the results. You can consider the offset table to be a set of three offset pairs.

If the two offsets in any pair are equal, the pair is empty and you can ignore it. Otherwise the pair identifies a run of characters whose glyphs are to be highlighted.

The offsets that `HiliteText` returns depend on the primary line direction as defined by the `SysDirection` global variable. If you change the value of `SysDirection`, `HiliteText` returns the offset that is meaningful according to the primary line direction for ambiguous offsets on the boundary of right-to-left and left-to-right text.

Special Considerations
The `HiliteText` function may move memory; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

InvokeStyleRunDirectionUPP

Calls your style run direction callback. (Deprecated in Mac OS X v10.4. Use ATSUI to handle style runs.)

```
Boolean InvokeStyleRunDirectionUPP (
   short styleRunIndex,
   void *dirParam,
   StyleRunDirectionUPP userUPP
);
```

Parameters
userUPP

Return Value
A `Boolean` value that indicates whether the callback was invoked successfully.

Discussion
You should not need to use the function `InvokeStyleRunDirectionUPP` as the system calls your style run direction callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`QuickdrawText.h`

MeasureJustified

Calculates, for text that is expanded, condensed, or scaled, the onscreen width in pixels from the left edge of the text segment to the glyph of the character. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void MeasureJustified (
   Ptr textPtr,
   long textLength,
   Fixed slop,
   Ptr charLocs,
   JustStyleCode styleRunPosition,
   Point numer,
   Point denom
);
```

Parameters

textPtr

A pointer to the memory location of the beginning of the text to be measured.

textLength

The number of bytes of text to be measured. The text length should equal the entire visible part of the text on a line, including trailing spaces if and only if they are displayed. Otherwise, the results for the last glyph on the line may be invalid.

slop

The amount of slop for the text to be drawn. A positive value extends the text segment; a negative value condenses the text segment.

The value of this parameter is the number of pixels by which the width of the text segment is to be changed, after the text has been scaled. The `slop` is a signed value that specifies how much the text is to be extended or condensed. The `slop` is derived from the calculations made using the proportion returned from the `PortionLine` function for a style run. To measure or draw text that is not to be extended or condensed, pass a `slop` value of 0.

charLocs

A pointer to an application-defined array of `textLength` + 1 integers.

styleRunPosition

The position on the line of this style run. The style run can be the only one on the line, the leftmost on the line, the rightmost on the line, or one between two other style runs.

This parameter specifies the position of the style run on the display line. It is used to determine the proportion of total slop to apply to a style run, measure or draw a line of justified text, identify where to break a line of text, and determine the caret position to mark an insertion point or highlight text.

The style run position parameter is meaningful only for those script systems that use intercharacter spacing for justification. For all other script systems, the parameter exists for future extensibility. Although the style run position parameter is not used, for example, for justifying text in the Roman script system, to allow for future compatibility, you should always specify the appropriate value for it for all calls that take it.

For those script systems that do use intercharacter spacing, space between style runs may be allocated differently depending upon whether the style run is leftmost, rightmost, or between two other style runs. For example, depending on the script system, if a style run occurs at the beginning or end of a line, extra space may not be added to the outer edge of the outermost glyph, whereas if a style run is interior to a line, all of the glyphs of the text may be treated the same: extra space is allocated to both sides of every glyph including those at either end of the style run.

The current implementations of simple script systems such as Roman and Cyrillic do not justify a line of text by changing the width of nonspace characters. Instead, they rely solely on the use of space characters: the same amount of extra width is added to (or subtracted from) every space whether the space is at the beginning or end of the line or interior to it.

See "Style Run Position Constants" (page 2951) for a list of the constants you can supply.

numer

A point giving the numerator for the horizontal and vertical scaling factors.

Both numer and denom are point values: numer specifies the numerator for the horizontal and vertical scaling factors, and denom specifies the denominator for the horizontal and vertical scaling factors. Together, these values specify the scaling factors for the text: `numer.v` over `denom.v` gives the vertical scaling (height), and `numer.h` over `denom.h` gives the horizontal scaling factors (width). You need to specify values for numer and denom even if you are not scaling the text. For unscaled text, you can specify scaling factors of 1, 1.

denom

A point giving the denominator for the horizontal and vertical scaling factors.

Discussion

The `MeasureJustified` function is similar to the `MeasureText` function, except that it is used to find the pixel location of a character's glyph in text that is expanded or condensed. The function calculates the onscreen pixel width of the glyph of each character, beginning from the left edge of the text segment, taking into account `slop` value, scaling, and style run position.

On return, the first element in the `charLocs` array contains 0 and the last element contains the total width of the text segment, when the primary line direction is left to right and the text is unidirectional. When the primary line direction is right to left and the text is unidirectional, the first element in the array contains the total width of the text segment, and the last element in the array contains 0. When the text is bidirectional, at a direction boundary, `MeasureJustified` selects the character whose direction maps to that of the primary line direction.

The `MeasureJustified` function returns the same results that an application would get if it called `CharToPixel` for each character with a direction parameter value of `hilite`. Using `MeasureJustified` to find the pixel location of a character's glyph is less efficient than using the `CharToPixel` function because the application must define the array pointed to by `charLocs`, and then walk the array after `MeasureText` returns the results.

The `MeasureJustified` function temporarily resets the space extra (`spExtra`) value, adding to the current value of the field, if any, the amount of extra space to be added to space characters in order to fully justify the text, based on calculations that take into account the slop value and all the text characteristics. On exit, `MeasureJustified` restores the original value.

Because `MeasureJustified` measures text in only the current font, style, and size, you need to call it once for each individual style run. For additional information about `MeasureJustified`, contact Developer Technical Support.

The `MeasureJustified` function works properly for text in all script systems. For 1-byte complex script systems, `MeasureJustified` calculates the widths of any ligatures, reversals, and compound characters that would need to be drawn.

Note that `textLength` is the number of bytes to be drawn, not the number of characters. Because 2-byte script systems also include characters consisting of only one byte, you should not simply multiply the number of characters by 2 to determine this value; the application must determine and specify the correct number of bytes.

Some 1-byte script system fonts may have zero-width characters, which are usually overlapping diacritical marks that typically follow the base character in memory. In this case, `MeasureJustified` measures both the glyph of the base character (the high-order, low-address byte) and the width of the diacritical mark. The `charLoc` array includes an entry for each, but both entries contain the same value.

For 1-byte complex script systems, `MeasureJustified` calculates the widths of any ligatures, reversals, compound characters, and character clusters that need to be drawn. For example, for an Arabic ligature, the entry that corresponds to the trailing edge of each character that is part of the ligature is the trailing edge of the entire ligature.

Special Considerations

The `MeasureJustified` function may move memory; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

MeasureText

Calculates the width of the character's glyph in pixels from the left edge of the text segment. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void MeasureText (
   short count,
   const void *textAddr,
   void *charLocs
);
```

Parameters

count

The number of bytes (as opposed to characters) to be measured. Because 2-byte script systems also include characters consisting of only one byte, do not simply multiply the number of characters by 2 to determine this value; you must determine and specify the correct number of bytes.

For 2-byte characters, the `charLocs` array contains two entries—one corresponding to each byte—but both entries contain the same pixel-width value.

textAddr

A pointer to the memory location of the beginning of the text to be measured. The value of `textAddr` must point directly to the first character whose glyph is to be measured.

charLocs

A pointer to an application-defined array of `count` + 1 integers. On return, the first element in the `charLocs` array contains 0 and the last element contains the total width of the text segment, when the primary line direction is left to right and the text is unidirectional.

When the primary line direction is right to left, and the text is unidirectional, the first element in the array contains the total width of the text segment, and the last element in the array contains 0. When the text is bidirectional, at a direction boundary, `MeasureText` selects the character whose direction maps to that of the primary line direction.

Discussion

Provides an array version of the `TextWidth` function. The `MeasureText` function calculates the onscreen pixel width of the glyph of each character, beginning from the left edge of the text segment. The function returns the same results that an application would get if it called `CharToPixel` for each character with a direction parameter value of `hilite`. Using `MeasureText` to find the pixel location of a character's glyph is less efficient than using the `CharToPixel` function because the application must define the array pointed to by `charLocs`, and then walk the array after `MeasureText` returns the results.

Because this function measures text in the font, style, and size of the current graphics port, you need to call it once for each individual style run in any line of text that contains multiple style runs.

Some fonts in 1-byte script systems may have zero-width characters, which are usually overlapping diacritical marks that typically follow the base character in memory. In this case, `MeasureText` measures both the glyph of the base character (the high-order, low-address byte) and the width of the diacritical mark. The `charLoc` array includes an entry for each, but both entries contain the same value.

For 1-byte complex script systems, `MeasureText` calculates the widths of any ligatures, reversals, compound characters, and character clusters that need to be drawn. For example, for an Arabic ligature, the entry that corresponds to the trailing edge of each character that is part of the ligature is the trailing edge of the entire ligature.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

NewStyleRunDirectionUPP

Creates a new universal procedure pointer (UPP) to a style run direction callback. (Deprecated in Mac OS X v10.4. Use ATSUI to handle style runs.)

```
StyleRunDirectionUPP NewStyleRunDirectionUPP (
   StyleRunDirectionProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `StyleRunDirectionUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`QuickdrawText.h`

PixelToChar

Returns the byte offset of a character in a style run, or part of a style run, whose onscreen glyph is nearest the place where the user clicked the mouse. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
short PixelToChar (
   Ptr textBuf,
   long textLength,
   Fixed slop,
   Fixed pixelWidth,
   Boolean *leadingEdge,
   Fixed *widthRemaining,
   JustStyleCode styleRunPosition,
   Point numer,
   Point denom
);
```

Parameters

textBuf

A pointer to the start of the text segment.

textLength

The length in bytes of the entire text segment pointed to by `textBuf`. The `PixelToChar` function requires the context of the complete text segment in order to determine the correct value.

slop

The amount of slop for the text to be drawn. A positive value extends the text segment; a negative value condenses the text segment.

The value of this parameter is the number of pixels by which the width of the text segment is to be changed, after the text has been scaled. The `slop` is a signed value that specifies how much the text is to be extended or condensed. The `slop` is derived from the calculations made using the proportion returned from the `PortionLine` function for a style run. To measure or draw text that is not to be extended or condensed, pass a `slop` value of 0.

pixelWidth

The screen location of the glyph associated with the character whose byte offset is to be returned. The screen location is measured in pixels beginning from the left edge of the text segment for which you call `PixelToChar`.

leadingEdge

Pointer to a Boolean flag that, upon completion of the call, is set to `TRUE` if the pixel location is on the leading edge of the glyph, and `FALSE` if the pixel location is on the trailing edge of the glyph. The leading edge is the left side if the direction of the character that the glyph represents is left-to-right (such as a Roman character), and the right side if the character direction is right-to-left (such as an Arabic or a Hebrew letter).

widthRemaining

Pointer to a location that, upon completion of the call, contains –1 if the pixel location (specified by the `pixelWidth` parameter) falls within the style run (represented by the `textLen` bytes starting at `textBuf`). Otherwise, the location contains the amount of pixels by which the input pixel location (`pixelWidth`) extends beyond the right edge of the text for which you called `PixelToChar`.

styleRunPosition

The position on the line of this style run. The style run can be the only one on the line, the leftmost on the line, the rightmost on the line, or one between two other style runs.

This parameter specifies the position of the style run on the display line. It is used to determine the proportion of total slop to apply to a style run, measure or draw a line of justified text, identify where to break a line of text, and determine the caret position to mark an insertion point or highlight text.

The style run position parameter is meaningful only for those script systems that use intercharacter spacing for justification. For all other script systems, the parameter exists for future extensibility. Although the style run position parameter is not used, for example, for justifying text in the Roman script system, to allow for future compatibility, you should always specify the appropriate value for it for all calls that take it.

For those script systems that do use intercharacter spacing, space between style runs may be allocated differently depending upon whether the style run is leftmost, rightmost, or between two other style runs. For example, depending on the script system, if a style run occurs at the beginning or end of a line, extra space may not be added to the outer edge of the outermost glyph, whereas if a style run is interior to a line, all of the glyphs of the text may be treated the same: extra space is allocated to both sides of every glyph including those at either end of the style run.

The current implementations of simple script systems such as Roman and Cyrillic do not justify a line of text by changing the width of nonspace characters. Instead, they rely solely on the use of space characters: the same amount of extra width is added to (or subtracted from) every space whether the space is at the beginning or end of the line or interior to it.

See "Style Run Position Constants" (page 2951) for a list of the constants you can supply.

numer

A point giving the numerator for the horizontal and vertical scaling factors.

Both `numer` and `denom` are point values: `numer` specifies the numerator for the horizontal and vertical scaling factors, and `denom` specifies the denominator for the horizontal and vertical scaling factors. Together, these values specify the scaling factors for the text: `numer.v` over `denom.v` gives the vertical scaling (height), and `numer.h` over `denom.h` gives the horizontal scaling factors (width). You need to specify values for `numer` and `denom` even if you are not scaling the text. For unscaled text, you can specify scaling factors of 1, 1.

denom

A point giving the denominator for the horizontal and vertical scaling factors.

Return Value
The byte offset of a character in a style run, or part of a style run, whose onscreen glyph is nearest the place where the user clicked the mouse.

Discussion
You can use the information that `PixelToChar` returns for highlighting, word selection, and identifying the caret position. The `PixelToChar` function returns a byte offset and a Boolean value that describes whether the pixel location is on the leading edge or trailing edge of the glyph where the mouse-down event occurred. When the pixel location falls on a glyph that corresponds to one or more characters that are part of the text segment, the `PixelToChar` function uses the direction of the character or characters to determine which side of the glyph is the leading edge. (A glyph can represent more than one character, for example, for a ligature. Generally, if a glyph represents more than one character, all of the characters have the same text direction.)

If the pixel location is on the leading edge, `PixelToChar` returns the byte offset of the character whose glyph is at the pixel location. (If the glyph represents multiple characters, it returns the byte offset of the first of these characters in memory.) If the pixel location is on the trailing edge, `PixelToChar` returns the byte offset of the first character in memory following the character or characters represented by the glyph. If the pixel location is on the trailing edge of the glyph that corresponds to the last character in the text segment, `PixelToChar` returns a byte offset equal to the length of the text segment.

When the pixel location is before the leading edge of the first glyph in the displayed text segment, `PixelToChar` returns a leading edge value of `FALSE` and the byte offset of the first character. When the pixel location is after the trailing edge of the last glyph in the displayed text segment, `PixelToChar` returns a leading edge value of `TRUE` and the next byte offset in memory, the one after the last character in the text segment. If the primary line direction is left to right, before means to the left of all of the glyphs for the characters in the text segment, and after means to the right of all these glyphs. If the primary line direction is right to left, before and after hold the opposite meanings.

You also use the value of the `leadingEdge` flag to help determine the value of the `direction` parameter to pass to `CharToPixel`, which you call to get the caret position. If the `leadingEdge` flag is `FALSE`, you base the value of the `direction` parameter on the direction of the character at the byte offset in memory that precedes the one that `PixelToChar` returns; if `leadingEdge` is `TRUE`, you base the value of the `direction` parameter on the direction of the character at the byte offset that `PixelToChar` returns. If there isn't a character at the byte offset, you base the value of the `direction` parameter on the primary line direction as determined by the `SysDirection` global variable.

You specify a value for `textLen` that is equal to the entire visible part of the style run on a line and includes trailing spaces if and only if they are displayed. They may not be displayed, for example, for the last style run in memory order that is part of the current line.

Be sure to pass the same values for `styleRunPosition` and the scaling factors (`numer` and `denom`) to `PixelToChar` that you pass to any of the other justification functions for this style run.

You pass `PixelToChar` a pointer to the byte offset of the character in the text buffer that begins the text segment or style run containing the character whose glyph is at the pixel location. If you do not know which style run on the display line contains the character whose glyph is at the pixel location, you can loop through the style runs until you find the one that contains the pixel location. If the style run contains the character, `PixelToChar` returns its byte offset. If it doesn't, you can use the `widthRemaining` parameter value to help determine which style run contains the glyph at the pixel location.

If you pass `PixelToChar` the pixel width of the display line, you can use the returned value of `widthRemaining` to calculate the length of a style run. The `widthRemaining` parameter contains the length in pixels from the end of the style run for which you call `PixelToChar` to the end of the display line, in this

case, if the style run for which you call it does not include the byte offset whose glyph corresponds to the pixel location. You subtract the returned `widthRemaining` value from the screen pixel width of the display line to get the style run's length.

To truncate a line of text, you can use `PixelToChar` to find the byte offset of the character where the line should be broken. To return the correct byte offset associated with the pixel location of a mouse-down event when the text belongs to a right-to-left script system, the `PixelToChar` function reorders the text. If right-to-left text is reordered when you use `PixelToChar` to determine where to break a line, it returns the wrong byte offset. To get the correct result, you must turn off reordering before you call `PixelToCha` r. Remember to restore reordering after you have determined where to break the line.

The `PixelToChar` function works with text in all script systems, and for text that is justified or not. For contextual script systems, `PixelToChar` takes into account the widths of any ligatures, reversals, and compound characters that were created when the text was drawn.

Because 2-byte script systems also include characters consisting of only one byte, you should not simply multiply the number of characters by 2 to determine this value; you must determine and specify the correct number of bytes.

Special Considerations

The `PixelToChar` function may move memory; do not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

PortionLine

Determines the correct proportion of extra space to apply to the specified style run in a line of justified text; that is, how to distribute the total slop value for a line among the style runs on that line. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
Fixed PortionLine (
   Ptr textPtr,
   long textLen,
   JustStyleCode styleRunPosition,
   Point numer,
   Point denom
);
```

Parameters

textPtr

A pointer to the style run.

textLen

The number of bytes in the text of the style run.

styleRunPosition

The position on the line of this style run. The style run can be the only one on the line, the leftmost on the line, the rightmost on the line, or one between two other style runs.

This parameter specifies the position of the style run on the display line. It is used to determine the proportion of total slop to apply to a style run, measure or draw a line of justified text, identify where to break a line of text, and determine the caret position to mark an insertion point or highlight text.

The style run position parameter is meaningful only for those script systems that use intercharacter spacing for justification. For all other script systems, the parameter exists for future extensibility. Although the style run position parameter is not used, for example, for justifying text in the Roman script system, to allow for future compatibility, you should always specify the appropriate value for it for all calls that take it.

For those script systems that do use intercharacter spacing, space between style runs may be allocated differently depending upon whether the style run is leftmost, rightmost, or between two other style runs. For example, depending on the script system, if a style run occurs at the beginning or end of a line, extra space may not be added to the outer edge of the outermost glyph, whereas if a style run is interior to a line, all of the glyphs of the text may be treated the same: extra space is allocated to both sides of every glyph including those at either end of the style run.

The current implementations of simple script systems such as Roman and Cyrillic do not justify a line of text by changing the width of nonspace characters. Instead, they rely solely on the use of space characters: the same amount of extra width is added to (or subtracted from) every space whether the space is at the beginning or end of the line or interior to it.

See "Style Run Position Constants" (page 2951) for a list of the constants you can supply.

numer

A point giving the numerator for the horizontal and vertical scaling factors.

Both `numer` and `denom` are point values: `numer` specifies the numerator for the horizontal and vertical scaling factors, and `denom` specifies the denominator for the horizontal and vertical scaling factors. Together, these values specify the scaling factors for the text: `numer.v` over `denom.v` gives the vertical scaling (height), and `numer.h` over `denom.h` gives the horizontal scaling factors (width). You need to specify values for `numer` and `denom` even if you are not scaling the text. For unscaled text, you can specify scaling factors of 1, 1.

denom

A point giving the denominator for the horizontal and vertical scaling factors.

Return Value
A number that represents the portion of the slop to be applied to the style run for which it is called.

Discussion
You use `PortionLine` in formatting a line of justified text. It helps you determine how to distribute the slop for a line among its style runs. When you know the total slop for a line of text, you need to determine what portion of it to attribute to each style run. To do this, you call the `PortionLine` function once for each style run on the line. The `PortionLine` function computes the portion of extra space for a style run, taking into account the font, size, style, and scaling factors of the style run. It returns a number that represents the portion of the slop to be applied to the style run for which it is called. You use the value that `PortionLine` returns to determine the percentage of slop that you should attribute to a style run.

To determine the percentage of slop to allocate to each style run, you compute what percentage each portion is of the sum of all portions. To determine the actual slop value in pixels for each style run, you apply the percentage to the total slop value. The following steps summarize this process:

1. Call `PortionLine` for each style run on the line.

2. Add the returned values together.

3. Calculate the percentage of the slop value for each style run using the ratio of the value returned by `PortionLine` for that style run and the total of the values returned for all of the style runs on the line.

4. Calculate the number of pixels to be added to each style run by multiplying the percentage of the slop for each style run by the total number of pixels.

Be sure to pass the same values for `styleRunPosition` and the scaling factors (`numer` and `denom`) to `PortionLine` that you pass to any of the other justification functions for this style run.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

SpaceExtra

Specifies the number of pixels by which to widen (or narrow) each space in a style run to be drawn in the current graphics port. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
void SpaceExtra (
   Fixed extra
);
```

Parameters

extra

The amount (in pixels or binary fractions of a pixel) to widen (or narrow) each space in a style run on a line.

Discussion
The `SpaceExtra` function sets the value of the extra space (`spExtra`) field in the current graphics port structure. The initial setting is `0`. You can pass a negative value for the extra parameter, but be careful not to narrow spaces so much that the text is unreadable. The value you specify is added to the width of each space character in the style run. For those script systems that do not use spaces, any value set in the extra space field is ignored. For those script systems that use spaces as delimiters, if you do not want to justify a line of text using `DrawJustified`, you can use the `SpaceExtra` function to set a fixed number of pixels to be added to each space character, then call `DrawText` or `DrawString`.

When you use the justification functions (`MeasureJustified`, `DrawJustified`) to measure or draw justified text, they temporarily reset the extra space value. They add to the current value of the field, if any, the amount of extra space to be added to space characters in the specified text in order to justify the text, based on calculations that take into account the slop value for the range of text and all of the text characteristics. On exit, these functions restore the original value.

For a color graphics port (`CGrafPort`), you can use `SpaceExtra` by itself or in conjunction with the `CharExtra` function to format a line of text in the 1-byte simple or 2-byte script systems. You should not use `CharExtra` for 1-byte complex script systems.

To ensure future compatibility and benefit from any enhancements, always use this function to modify the `spExtra` field, rather than directly change the field value.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

StandardGlyphs

This obsolete function doesn't do anything in Mac OS X. (Deprecated in Mac OS X v10.4. Use ATSUI to render Unicode text.)

Not recommended.

```
OSStatus StandardGlyphs (
   void *dataStream,
   ByteCount size
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

StdText

Draws text from an arbitrary structure in memory. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void StdText (
   short count,
   const void *textAddr,
   Point numer,
   Point denom
);
```

Parameters

count
The number of bytes of text to draw.

textAddr
A memory structure containing the text to draw.

numer
Scaling numerator.

denom
Scaling denominator.

Discussion
This is QuickDraw's standard low-level function for drawing text. The `StdText` function draws text from the arbitrary structure in memory specified by the `textBuf` parameter, starting from the first byte and continuing for the number of bytes specified in the `byteCount` parameter. The `numer` and `denom` parameters specify the scaling factor: `numer.v` over `denom.v` gives the vertical scaling, and `numer.h` over `denom.h` gives the horizontal scaling factor.

You should only call this low-level function from your customized QuickDraw functions.

Special Considerations

The `StdText` function may move or purge memory blocks in the application heap; do not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

stdtext

Draws text from an arbitrary structure in memory. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

Modified

```
void stdtext (
   short count,
   const void *textAddr,
   const Point *numer,
   const Point *denom
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

StdTxMeas

Measures the width of scaled or unscaled text. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
short StdTxMeas (
   short byteCount,
   const void *textAddr,
   Point *numer,
   Point *denom,
   FontInfo *info
);
```

Parameters

byteCount

The number of bytes to be counted.

textAddr

A pointer to the beginning of the text in memory.

numer

Pointer to a point giving the numerator for the horizontal and vertical scaling factors. For this function, `numer` and `denom` are reference parameters. On output, these parameters contain additional scaling to be applied to the text.

Both `numer` and `denom` are point values: `numer` specifies the numerator for the horizontal and vertical scaling factors, and `denom` specifies the denominator for the horizontal and vertical scaling factors. Together, these values specify the scaling factors for the text: `numer.v` over `denom.v` gives the vertical scaling (height), and `numer.h` over `denom.h` gives the horizontal scaling factors (width). You need to specify values for `numer` and `denom` even if you are not scaling the text. For unscaled text, you can specify scaling factors of 1, 1.

denom

Pointer to a point giving the denominator for the horizontal and vertical scaling factors.

info

Pointer to a font information structure that describes the current font.

Return Value

The width of the text stored in memory, beginning with the first character at `textAddr` and continuing for `byteCount` bytes.

Discussion

The `StdTxMeas` function is a QuickDraw bottleneck function that the QuickDraw text-measuring functions use extensively. The `StdTxMeas` function returns the width of the text stored in memory beginning with the first character at `textAddr` and continuing for `byteCount` bytes. You can call the `StdTxMeas` function directly, for example, to measure text that you want to explicitly scale, but not justify. You can also use `StdTxMeas` to get the font metrics for scaled text in order to determine the line height, instead of using `GetFontInfo`, which doesn't support scaling.

The high-level QuickDraw text functions provide most of the functionality needed to measure and draw text. However, if you need to call `StdTxMeas` directly, you must first check the graphics port `grafProcs` field to determine whether the bottleneck functions have been customized, and if so, you must call the customized function instead of the standard one. The bottleneck functions are always customized for printing.

If the `grafProcs` field contains `NULL`, the standard bottleneck functions have not been customized. If the `grafProcs` field contains a pointer, the standard bottleneck functions have been replaced by customized ones. This pointer (of type `QDProcPtr`) points to a `QDProc` structure, which contains fields that point to the bottleneck function to be used for a specific drawing function. If the standard bottleneck function has been customized, your application needs to use the customized function indicated by the `QDProc` structure field.

On input, you need to specify values for `numer` and `denom`, even if you are not scaling the text. You can specify 1,1 scaling factors, in this case, so that no scaling is applied. On return, `numer` and `denom` contain the additional scaling to be applied to the text.

The `StdTxtMeas` function returns output scaling factors that you need to apply to the text to get the right measurement if the Font Manager was not able to fully satisfy the scaling request. You can use the Toolbox Utilities' `FixRound` and `FixRatio` functions to help with this process.

The `StdTxMeas` function gives the correct results for all script systems. The `byteCount` parameter is the number of bytes of the text to be drawn, not characters. When specifying this value, consider that 2-byte script systems also include characters consisting of only one byte.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

StringWidth

Returns the length, in pixels, of the specified text string. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
short StringWidth (
   ConstStr255Param s
);
```

Parameters

s

A Pascal string containing the text to be measured.

Return Value
The length, in pixels, of the specified text string.

Discussion
You should not call `StringWidth` to measure scaled text. Although `StringWidth` takes into account the graphics port structure settings, it does not accept scaling parameters, and therefore cannot determine the correct text width result for text to be drawn using scaling factor parameters.

If you specify values in the graphics port `spExtra` or `chExtra` fields to change the width of space or nonspace characters, `StringWidth` takes these values into account.

Because this function measures text in the font, style, and size of the current graphics port, you need to call it once for each individual style run in any line of text that contains multiple style runs.

The `StringWidth` function works with all script systems.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`QuickdrawText.h`

StyledLineBreak

Returns the proper location to break a line of text, taking into account script and language considerations, making use of tables in the string-manipulation (`'itl2'`) resource in its computations. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
StyledLineBreakCode StyledLineBreak (
   Ptr textPtr,
   SInt32 textLen,
   SInt32 textStart,
   SInt32 textEnd,
   SInt32 flags,
   Fixed *textWidth,
   SInt32 *textOffset
);
```

Parameters

`textPtr`
A pointer to the beginning of a script run on the current line to be broken.

`textLen`
The number of bytes in the script run on the current line to be broken.

`textStart`
A byte offset to the beginning of a style run within the script run.

When used with unformatted text, `textStart` can be 0, and `textEnd` is identical to `textLen`. With styled text, the interval between `textStart` and `textEnd` specifies a style run. The interval between `textPtr` and `textLen` specifies a script run. Note that the style runs in `StyledLineBreak` must be traversed in memory order, not in display order.

`textEnd`
A byte offset to the end of the style run within the script run.

`flags`
Reserved for future expansion; must be 0.

`textWidth`
A pointer to the maximum length of the displayed line in pixels. `StyledLineBreak` decrements this value for its own use. You are responsible for setting it before your first call to `StyledLineBreak` for a line.

`StyledLineBreak` automatically decrements the `textWidth` variable by the width of the style run for use on the next call. You need to set the value of `textWidth` before calling it to process a line.

textOffset

A pointer to the text offset value, which must be nonzero on your first call to `StyledLineBreak` for a line, and zero for subsequent calls to `StyledLineBreak` for that line. This value allows `StyledLineBreak` to differentiate between the first and subsequent calls, which is important when a long word is found (as described below).

The `textOffset` parameter must be nonzero for the first call on a line and zero for each call to the function on the line. This allows `StyledLineBreak` to act differently when a long word is encountered: if the word is in the first style run on the line, `StyledLineBreak` breaks the line on a character boundary within the word; if the word is in a subsequent style run on the line, `StyledLineBreak` breaks the line before the start of the word.

On output, `textOffset` is the count of bytes from `textPtr` to the location in the text string where the line break is to occur. When `StyledLineBreak` finds a line break, it sets the value of `textOffset` to the count of bytes that can be displayed starting at `textPtr`.

When `StyledLineBreak` is called for the second or subsequent style runs within a script run, the `textOffset` value at exit may be less than the `textStart` parameter (that is, it may specify a line break before the current style run).

Return Value
Indicates whether the function broke on a word boundary or a character boundary, or if the width extended beyond the edge of the text. See "Style Line Break Values" (page 2950) for a list of the constants that can be returned.

Discussion
The function `StyledLineBreak` breaks the line on a word boundary if possible and allows for multiscript runs and style runs on a single line.

Use the `StyledLineBreak` function when you are laying out lines in an environment that may include text from multiple scripts. To use this function, you need to understand how QuickDraw draws text.

You can only use the `StyledLineBreak` function when you have organized your text in script runs and style runs within each script run. This type of text organization is used by most text-processing applications that allow for multiscript text. Use this function when you are displaying text in a screen area to determine the best place to break each displayed line.

What you do is iterate through your text, a script run at a time starting from the first character past the end of the previous line. Use `StyledLineBreak` to check each style run in the script run (in memory order) until the function determines that it has arrived at a line break. As you loop through each style run, before calling `StyledLineBreak`, you must set the text values in the graphics port structure that are used by QuickDraw to measure the text. These include the font, font size, and font style of the style run.

If the current style run is included in a contiguous sequence of other style runs of the same script, then `textPtr` should point to the start of the first style run of the same script on the line, and `textLen` should include the last style run of the same script on the line. This is because word boundaries can extend across style runs, but not across script runs.

Although the offsets are in long integer values and the widths are in fixed values for future extensions, in the current version the long integer values are restricted to the integer range, and only the integer portion of the widths is used.

`StyledLineBreak` always chooses a line break for the last style run on the line in memory order as if all whitespace in that style run would be stripped. The `VisibleLength` function, which is a QuickDraw function used to eliminate trailing spaces from a style run before drawing it, can be called for the style run that is at

the display end of a line. This leads to a potential conflict when both functions are used with mixed-directional text: if the end of a line in memory order actually occurs in the middle of the displayed line, `StyledLineBreak` assumes that the whitespace is stripped from that run, but `VisibleLength` does not strip the characters.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

TextFace

Sets the style of the font in which the text is to be drawn in the current graphics port. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void TextFace (
   StyleParameter face
);
```

Parameters

face

The style for text to be drawn in the current graphics port.

Discussion
The `TextFace` function sets the value for the style of the font in the text face (`txFace`) field of the current graphics port. The `Style` data type allows you to specify a set of one or more of the following predefined constants: `bold`, `italic`, `underline`, `outline`, `shadow`, `condense`, and `extend`. In Pascal, you specify the constants within square brackets. For example:

```
TextFace([bold]);
{bold}TextFace([bold,italic]);
{bold and italic}
```

The style is set to the empty set (`[]`) by default, which specifies plain.

To ensure future compatibility and benefit from any enhancements, always use this function to modify the `txFace` field, rather than directly change the field value.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

TextFont

Sets the font of the current graphics port in which the text is to be rendered. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void TextFont (
   short font
);
```

Parameters

font

The font family ID. The initial font family ID is `0`, which represents the system font. The value that you specify for this field is either an integer or a constant. The range of integers currently defined are from `0` to `32767`. Currently, negative font family IDs are not supported, although they may be supported in the future.

The system font and application font have different font IDs and sizes on various script systems. However, the special designators `0` and `1` always map to the system font and the application font for the system script, respectively.

Discussion
The `TextFont` function sets the value of the graphics port text font (`txFont`) field. To ensure future compatibility and benefit from any enhancements, always use this function to modify the `txFont` field, rather than directly change the field value.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`QuickdrawText.h`

TextMode

Sets the transfer mode for drawing text in the current graphics port. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void TextMode (
   short mode
);
```

Parameters

mode

The transfer mode to be used to draw the text.

Discussion
The `TextMode` function sets the transfer mode in the graphics port `txMode` field. The transfer mode determines the interplay between what an application is drawing (the source) and what already exists on the display device (the destination), resulting in the text display.

There are two basic kinds of modes: pattern (`pat`) and source (`src`). Source is the kind that you use for drawing text. There are four basic Boolean operations: `Copy`, `Or`, `Xor`, and `Bic` (bit clear), each of which has an inverse variant in which the source is inverted before the transfer, yielding eight operations in all. Original QuickDraw supports these eight transfer modes. Color QuickDraw enables your application to achieve color

effects within those basic transfer modes, and offers an additional set of transfer modes that perform arithmetic operations on the RGB values of the source and destination pixels. Other transfer modes are `grayishTextOr`, `transparent` mode, and text mask mode.

To ensure future compatibility and benefit from any enhancements, always use this function to modify the `txMode` field, rather than directly change the field value.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`QuickdrawText.h`

TextSize

Sets the font size for text drawn in the current graphics port to the specified number of points. (Deprecated in Mac OS X v10.4. Use ATSUI or Quartz instead.)

```
void TextSize (
   short size
);
```

Parameters

size

The font size in points (0 to 32,767). The initial setting is 0, which specifies that the font size of the system font (normally 12 points) is to be used.

Discussion
The `TextSize` function sets the font size in the text size (`txSize`) field of the current graphics port structure. To ensure future compatibility and benefit from any enhancements, always use this function to modify the `txSize` field, rather than directly change the field value.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`QuickdrawText.h`

TextWidth

Returns the length, in pixels, of the specified text. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
short TextWidth (
   const void *textBuf,
   short firstByte,
   short byteCount
);
```

Parameters

textBuf

A pointer to a buffer that contains the text to be measured.

firstByte

An offset from `textBuf` to the first byte of the text to be measured.

byteCount

The number of bytes of text to be measured.

Return Value

The length, in pixels, of the specified text.

Discussion

You can use `TextWidth` to measure the screen pixel width of any text segment that has uniform character attributes. You can use it to measure the style runs in a line of text, whether you intend to draw the line using `DrawText` or `DrawJustified`. The `TextWidth` function takes into account the character attributes set in the graphics port. If you change any of these attributes after determining the text width but before actually drawing the text, the predetermined width may not be correct. For a space character, `TextWidth` also includes the effect of `SpaceExtra`. For a nonspace character, `TextWidth` includes the effect of `CharExtra`.

Because this function measures text in the font, style, and size of the current graphics port, you need to call it once for each individual style run in any line of text that contains multiple style runs.

The `TextWidth` function works with text in all script systems because the script management system modifies the function if necessary to give the proper results.

To draw justified lines of text that include multiple style runs, you calculate the amount of extra pixels, or slop, that remains to be distributed throughout the line. This process entails measuring the screen pixel width of each style run on the line: you can use `TextWidth` for this purpose.

For 1-byte complex script systems, `TextWidth` calculates the widths of any ligatures, reversals, and compound characters that need to be drawn.

Note that `byteCount` is the number of bytes to be measured, not the number of characters. Because 2-byte script systems also include characters consisting of only one byte, you should not simply multiply the number of characters by 2 to determine this value; you must determine and specify the correct number of bytes.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

TruncString

Ensures that a Pascal string fits into the specified pixel width, by truncating the string as necessary. This function makes use of the current script and font. (Deprecated in Mac OS X v10.4. Use CFString instead.)

```
short TruncString (
   short width,
   Str255 theString,
   TruncCode truncWhere
);
```

Parameters

width

The number of pixels in which the string must be displayed in the current script and font.

theString

The Pascal string to be displayed. On output, contains a version of the string that has been truncated (if necessary) to fit in the number of pixels specified by `width`.

truncWhere

A constant that indicates where the string should be truncated. If you supply the `truncEnd` value, characters are truncated off the end of the string. If you supply the `truncMiddle` value, characters are truncated from the middle of the string; this is useful when displaying pathnames.

See "Truncation Positions" (page 2953) for a list of the constants you can supply.

Discussion

The `TruncString` function ensures that a Pascal string fits into the pixel width specified by the `width` parameter by modifying the string, if necessary, through truncation. `TruncString` uses the font script to determine how to perform truncation. If truncation occurs, `TruncString` inserts a truncation indicator, which is the ellipsis (…) in the Roman script system. You can specify which token to use for indicating truncation as the `tokenEllipsis` token type in the untoken table of a tokens (`'itl4'`) resource.

To determine the width of a string in the current font and script, use the QuickDraw `StringWidth` function.

Special Considerations

`TruncString` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

TruncText

Ensures that a text string fits into the specified pixel width, by truncating the string as necessary. This function makes use of the current script and font. (Deprecated in Mac OS X v10.4. Use CFString instead.)

```
short TruncText (
   short width,
   Ptr textPtr,
   short *length,
   TruncCode truncWhere
);
```

Parameters

width

The number of pixels in which the text string must be displayed in the current script and font.

textPtr

A pointer to the text string to be truncated. The text string can be up to 32 KB long.

length

On input, a pointer to a value containing the length, in bytes, of the text string to be truncated. On output, this value is updated to reflect the length of the (possibly) truncated text.

truncWhere

A constant that indicates where the text string should be truncated. You must set this parameter to one of the constants `truncEnd` or `truncMiddle`. If you supply the `truncEnd` value, characters are truncated off the end of the string. If you supply the `truncMiddle` value, characters are truncated from the middle of the string; this is useful when displaying pathnames.

See "Truncation Positions" (page 2953) for a list of the constants you can supply.

Discussion

You can use the `TruncText` function to ensure that a string defined by a pointer and a byte length fits into the specified pixel width, by truncating the string in a manner dependent on the font script.

`TruncText` uses the font script to determine how to perform truncation. If truncation occurs, `TruncText` inserts a truncation indicator which is the ellipsis (…) in the Roman script system. You can specify which token to use for indicating truncation as the `tokenEllipsis` token type in the untoken table of a tokens resource.

To determine the width of a string in the current font and script, use the QuickDraw `StringWidth` function.

Special Considerations

`TruncText` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

VisibleLength

Calculates the length, in bytes, of a given text segment, excluding trailing white space. (Deprecated in Mac OS X v10.4. Use ATSUI instead.)

```
long VisibleLength (
   Ptr textPtr,
   long textLength
);
```

Parameters

textPtr

A pointer to a text string.

textLength

The number of bytes in the text segment.

Return Value

The length, in bytes, of a given text segment, excluding trailing white space.

Discussion

The `VisibleLength` function determines how much of a style run to display, without displaying trailing spaces. You call `VisibleLength` for the last style run of a line in memory order. The last style run in memory order of the text constituting the line is not always the last style run in display order. For a line of unidirectional left-to-right text, the last style run in memory order is the rightmost style run in display order. For a line of unidirectional right-to-left text, the last style run in memory order is the leftmost style run in display order. However, if the text contains mixed directions, the last style run in memory order may be an interior style run in display order.

The text justification functions do not automatically exclude trailing spaces, so you pass them the value that `VisibleLength` returns as the length of the last style run in memory order.

The `VisibleLength` function behaves differently for various script systems. For simple script systems, such as Roman and Cryllic, and for 2-byte script systems, `VisibleLength` does not include in the byte count it returns trailing spaces that occur at the display end of the text segment. For 2-byte script systems, `VisibleLength` does not count them, whether they are 1-byte or 2-byte space characters.

For 1-byte complex script systems, `VisibleLength` does not include in the byte count that it returns spaces whose character direction is the same as the primary line direction. For 1-byte complex script systems that support bidirectional text, Roman spaces take on a character direction based on the primary line direction. If the Roman spaces then fall at the end of the text, `VisibleLength` does not include them in the returned byte count.

The purpose of `VisibleLength` is to trim off white space at the display end of the line. The `VisibleLength` function does not eliminate the white space by removing its character code from memory. Rather, it does not include white space characters in the count that it returns as the length of the range of text for which you call it.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`QuickdrawText.h`

Callbacks

StyleRunDirectionProcPtr

Defines a pointer to a style run direction callback function that calculates, for a style run identified by number, the direction of that style run.

```
typedef Boolean (*StyleRunDirectionProcPtr)
(
    short styleRunIndex,
    void * dirParam
);
```

If you name your function `MyStyleRunDirectionProc`, you would declare it like this:

```
Boolean StyleRunDirectionProcPtr (
    short styleRunIndex,
    void * dirParam
);
```

Parameters

styleRunIndex

A value that identifies the style run whose direction is needed.

dirParam

A pointer to an application-defined parameter block that contains the font and script information for each style run in the text. The contents of this parameter block are used to determine the direction of the style run. Because of the relationship between the font family ID and the script code, the font family ID can be used to determine the text direction.

Return Value

A Boolean value that is `TRUE` for right-to-left text direction, `FALSE` for left-to-right.

Discussion

To fill the ordering array (type `FormatOrder`) for style runs on a line, the `GetFormatOrder` function calls `MyStyleRunDirectionCallback` for each style run numbered from `firstFormat` to `lastFormat`. `GetFormatOrder` passes `MyStyleRunDirectionCallback` a number identifying the style run in storage order, and a pointer to the parameter information block, `dirParam`, that contains the font and style information for the style run. Given `dirParam` and a style run identifier, the application-defined `MyStyleRunDirectionCallback` function should be able to determine the style run direction.

You should store your style run information in a way that makes it convenient for `MyStyleRunDirectionCallback`. One obvious way to do this is to declare a structure type for style runs that allows you to save things like font style, font family ID, script number, and so forth. You then can store these structures in an array. When the time comes for `GetFormatOrder` to fill the ordering array, `MyStyleRunDirectionCallback` can consult the style run array for direction information for each of the numbered style runs in turn.

For more information, see `GetFormatOrder` (page 2920).

When you provide the Component Manager with a pointer to your function, you should use a universal procedure pointer (UPP). The definition of the UPP data type for your file identification function is as follows:

```
typedef (StyleRunDirectionProcPtr) StyleRunDirectionUPP;
```

Before using your style run direction callback function, you must first create a new universal procedure pointer to it, using the `NewStyleRunDirectionUPP` function, as shown here:

```
StyleRunDirectionUPP MyStyleRunDirectionUPP;

MyStyleRunDirectionUPP = StyleRunDirectionUPP(&MyStyleRunDirectionCallback)
```

You then pass `MyStyleRunDirectionUPP` to the function `GetFormatOrder`. If you wish to call your own callback function, you can use the `InvokeStyleRunDirectionUPP` function:

```
direction = InvokeStyleRunDirectionUPP(styleRunIndex, &paramInfo,
MyStyleRunDirectionUPP)
```

When you are finished using your callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeStyleRunDirectionUPP` function.

```
DisposeStyleRunDirectionUPP(MyStyleRunDirectionUPP);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawText.h`

Data Types

FontInfo

Contains font metric information.

```
struct FontInfo {
    short ascent;
    short descent;
    short widMax;
    short leading;
};
typedef struct FontInfo FontInfo;
```

Fields

`ascent`
The measurement, in pixels, from the baseline to the ascent line of the font.

`descent`
The measurement, in pixels, from the baseline to the descent line of the font.

`widMax`
The width, in pixels, of the largest glyph in the font.

`leading`
The measurement, in pixels, from the descent line to the ascent line below it.

Discussion
The `FontInfo` data type defines a font information structure. The `GetFontInfo` (page 2919) function uses the font information structure to return measurement information based on the font of the current graphics port. If the current font has an associated font, as do Arabic and Hebrew, `GetFontInfo` returns information

based on both fonts. The font information structure contains the ascent, the descent, the width of the largest glyph, and the leading for a given font. The `StdTxMeas` (page 2934) function also uses a structure of type `FontInfo` to return information about the current font.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawText.h`

FormatOrder

Contains and array of display orders for style runs.

```
typedef  FormatOrder[1];
```

Discussion
The `GetFormatOrder` (page 2920) function fills the supplied format order array with the display order of each style run.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawText.h`

StyleRunDirectionUPP

Defines a universal procedure pointer to a style run direction callback.

```
typedef StyleRunDirectionProcPtr StyleRunDirectionUPP;
```

Discussion
For more information, see the description of the `StyleRunDirectionProcPtr` (page 2946) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawText.h`

Constants

Caret Direction Constants

Specify a caret position.

```
enum {
    leftCaret = 0,
    rightCaret = -1,
    kHilite = 1
};
```

Constants

`leftCaret`
Place caret for left-to-right text direction.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`rightCaret`
Place caret for right-to-left text direction.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`kHilite`
Specifies that the caret position should be determined according to the primary line direction, based on the value of `SysDirection`.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

Discussion
You can use these constants to specify a value for `direction`, as used in the `CharToPixel` (page 2911) function.

Truncation Status Values

Returned as result codes for the functions `TruncString` and `TruncText`.

```
enum {
    notTruncated = 0,
    truncated = 1,
    truncErr = -1,
    smNotTruncated = 0,
    smTruncated = 1,
    smTruncErr = -1
};
```

Constants

`notTruncated`
Specifies that truncation is not necessary.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`truncated`
Specifies that truncation was performed.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`truncErr`
Specifies a general error occurred.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smNotTruncated`
Specifies that truncation is not necessary. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smTruncated`
Specifies that truncation was performed. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smTruncErr`
Specifies a general error occurred. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

Style Line Break Values

Specify a line break.

```
typedef SInt8 StyledLineBreakCode;
enum {
    smBreakWord = 0,
    smBreakChar = 1,
    smBreakOverflow = 2
};
```

Constants

`smBreakWord`
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smBreakChar`
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smBreakOverflow`
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

Obsolete Caret Placement Values

Specify where to place a caret.

```
enum {
    smLeftCaret = 0,
    smRightCaret = -1,
    smHilite = 1
};
```

Constants

`smLeftCaret`
Specifies to place caret for left block. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smRightCaret`
Specifies to place caret for right block. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smHilite`
Specifies the direction is `TESysJust`. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

Style Run Position Constants

Specify style run positions.

```
typedef short JustStyleCode;
enum {
    onlyStyleRun = 0,
    leftStyleRun = 1,
    rightStyleRun = 2,
    middleStyleRun = 3,
    smOnlyStyleRun = 0,
    smLeftStyleRun = 1,
    smRightStyleRun = 2,
    smMiddleStyleRun = 3
};
```

Constants

`onlyStyleRun`
This is the only style run on the line.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`leftStyleRun`
This is the leftmost of multiple style runs on the line.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`rightStyleRun`
This is the rightmost of multiple style runs on the line
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`middleStyleRun`
The line and this one is interior: neither leftmost nor rightmost.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawText.h`.

`smOnlyStyleRun`
This is obsolete.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawText.h`.

`smLeftStyleRun`
This is obsolete.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawText.h`.

`smRightStyleRun`
This is obsolete.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawText.h`.

`smMiddleStyleRun`
This is obsolete.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawText.h`.

Discussion
Use one of the following constants (defined as type `JustStyleCode`) in the `styleRunPosition` parameter for `PortionLine` (page 2930) `DrawJustified` (page 2915), `MeasureJustified` (page 2923), `CharToPixel` (page 2911), and `PixelToChar` (page 2927).

txFlag Constants

Specify constants for `txFlags`.

```
enum {
    tfAntiAlias = 1 << 0,
    tfUnicode = 1 << 1
};
```

Constants
`tfAntiAlias`
Available in Mac OS X v10.0 and later.

Declared in `QuickdrawText.h`.

`tfUnicode`
Available in Mac OS X v10.0 and later.

Declared in `QuickdrawText.h`.

Discussion
These used to be the pad field after `txFace`.

Truncation Positions

Specify where to truncate a string.

```
typedef  TruncCode;
enum {
    truncEnd = 0,
    truncMiddle = 0x4000,
    smTruncEnd = 0,
    smTruncMiddle = 0x4000
};
```

Constants

`truncEnd`
Truncate at end.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`truncMiddle`
Truncate in middle.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smTruncEnd`
Truncate at end. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

`smTruncMiddle`
Truncate in middle. This is obsolete.
Available in Mac OS X v10.0 and later.
Declared in `QuickdrawText.h`.

Document Revision History

This table describes the changes to *Application Services Framework Reference*.

Date	Notes
2007-10-31	Added Core Printing. Removed Launch Services.
2007-05-11	Updated with a new document for Mac OS X v10.5.
2006-05-23	First publication of this content as a collection of previously published documents.

Document Revision History

Index

Numerals

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y