Application Kit Functions Reference

Cocoa > Objective-C Language

2008-11-19

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

Simultaneously published in the United States and Canada.

Contents

Application Kit Functions Reference

Framework:	AppKit/AppKit.h
Declared in	NSAccessibility.h NSApplication.h NSCell.h NSEvent.h NSFont.h NSGraphics.h NSInterfaceStyle.h NSKeyValueBinding.h NSOpenGL.h NSPanel.h NSPasteboard.h

Overview

This document describes functions and function-like macros defined in the Application Kit framework.

Functions by Task

Accessibility

Additional information on accessibility can be found in `NSAccessibility`.

`NSAccessibilityActionDescription` (page 12)
Returns a standard description for an action.

`NSAccessibilityPostNotification` (page 13)
Sends a notification to any observing assistive applications.

`NSAccessibilityRaiseBadArgumentException` (page 13)
Raises an error if the parameter is the wrong type or has an illegal value

`NSAccessibilityRoleDescription` (page 13)
Returns a standard description for a role and subrole.

`NSAccessibilityRoleDescriptionForUIElement` (page 14)
Returns a standard role description for a user interface element.

`NSAccessibilityUnignoredChildren` (page 15)
Returns a list of unignored accessibility objects, descending the hierarchy if necessary.

`NSAccessibilityUnignoredChildrenForOnlyChild` (page 15)
Returns a list of unignored accessibility objects, descending the hierarchy if necessary.

`NSAccessibilityUnignoredDescendant` (page 16)
Returns an unignored accessibility object, descending the hierarchy if necessary.

`NSAccessibilityUnignoredAncestor` (page 14)
Returns an unignored accessibility object, ascending the hierarchy if necessary.

Applications

Additional information on `NSApplication` can be found in *NSApplication Class Reference*.

`NSApplicationLoad` (page 16)
Startup function to call when running Cocoa code from a Carbon application.

`NSApplicationMain` (page 17)
Called by the main function to create and run the application.

`NSPerformService` (page 45)
Programmatically invokes a Services menu service.

`NSRegisterServicesProvider` (page 52)
Registers a service provider.

`NSSetShowsServicesMenuItem` (page 56)
Specifies whether an item should be included in Services menus.

`NSShowsServicesMenuItem` (page 57)
Specifies whether a Services menu item is currently enabled.

`NSUnregisterServicesProvider` (page 58)
Unregisters a service provider.

`NSUpdateDynamicServices` (page 58)
Causes the services information for the system to be updated.

Events

`NSEventMaskFromType` (page 36)
Returns the event mask for the specified type.

Fonts

`NSConvertGlyphsToPackedGlyphs` (page 23)
Prepares a set of glyphs for processing by character-based routines.

Graphics

`NSCopyBits` (page 23)
Copies a bitmap image to the location specified by a destination point.

`NSCountWindows` (page 23)
Counts the number of onscreen windows.

`NSCountWindowsForContext` (page 24)
Counts the number of onscreen windows belonging to a particular application.

`NSDisableScreenUpdates` (page 25)
Disables screen updates.

`NSEnableScreenUpdates` (page 36)
Enables screen updates

`NSDottedFrameRect` (page 25)
Draws a bordered rectangle.

`NSDrawBitmap` (page 26)
Draws a bitmap image.

`NSDrawButton` (page 27)
Draws a gray-filled rectangle representing a user-interface button.

`NSDrawDarkBezel` (page 28)
Draws a dark gray-filled rectangle with a bezel border.

`NSDrawGrayBezel` (page 29)
Draws a gray-filled rectangle with a bezel border.

`NSDrawGroove` (page 29)
Draws a gray-filled rectangle with a groove border.

`NSDrawLightBezel` (page 30)
Draws a white-filled rectangle with a bezel border.

`NSDrawThreePartImage` (page 32)
Draws a three-part tiled image.

`NSDrawNinePartImage` (page 30)
Draws a nine-part tiled image.

`NSDrawTiledRects` (page 33)
Draws rectangles with borders.

`NSDrawColorTiledRects` (page 28)
Draws a colored bordered rectangle.

`NSDrawWhiteBezel` (page 35)
Draws a white-filled rectangle with a bezel border.

`NSDrawWindowBackground` (page 35)
Draws the window's default background pattern into the specified rectangle of the currently focused view.

`NSEraseRect` (page 36)
Erases the specified rect by filling it with white.

`NSFrameRect` (page 37)
Draw a bordered rectangle.

`NSFrameRectWithWidth` (page 38)
Draw a bordered rectangle.

`NSFrameRectWithWidthUsingOperation` (page 38)
Draw a bordered rectangle using the specified compositing operation.

NSGetWindowServerMemory (page 41)
Returns the amount of memory being used by a context.

NSHighlightRect (page 42)
Highlights the specified rect by filling it with white.

NSReadPixel (page 46)
Reads the color of the pixel at the specified location.

NSRectClip (page 47)
Modifies the current clipping path by intersecting it with the passed rect.

NSRectClipList (page 48)
Modifies the current clipping path by intersecting it with the passed rect.

NSRectFill (page 48)
Fills the passed rectangle with the current color.

NSRectFillList (page 49)
Fills the rectangles in the passed list with the current fill color.

NSRectFillListWithColors (page 50)
Fills the rectangles in the passed list with the passed list of colors.

NSRectFillListWithGrays (page 51)
Fills the rectangles in the passed list with the passed list of grays.

NSRectFillListUsingOperation (page 49)
Fills the rectangles in a list using the current fill color and specified compositing operation.

NSRectFillListWithColorsUsingOperation (page 50)
Fills the rectangles in a list using the specified colors and compositing operation.

NSRectFillUsingOperation (page 52)
Fills a rectangle using the current fill color and the specified compositing operation.

NSSetFocusRingStyle (page 56)
Specifies how a focus ring will be drawn.

NSShowAnimationEffect (page 56)
Runs a system animation effect.

NSWindowList (page 58)
Gets information about onscreen windows.

NSWindowListForContext (page 59)
Gets information about an application's onscreen windows.

Graphics-Window Depth

NSAvailableWindowDepths (page 17)
Returns the available window depth values.

NSBestDepth (page 21)
Attempts to return a window depth adequate for the specified parameters.

NSBitsPerPixelFromDepth (page 22)
Returns the bits per pixel for the specified window depth.

NSBitsPerSampleFromDepth (page 22)
Returns the bits per sample for the specified window depth.

`NSColorSpaceFromDepth` (page 22)
Returns the name of the color space corresponding to the passed window depth.

`NSNumberOfColorComponents` (page 44)
Returns the number of color components in the specified color space.

`NSPlanarFromDepth` (page 46)
Returns whether the specified window depth is planar.

Interface Styles

`NSInterfaceStyleForKey` (page 43)
Returns an interface style value for the specified key and responder.

Key Value Bindings

`NSIsControllerMarker` (page 43)
Tests whether a given object is special marker object used for indicating the state of a selection in relation to a key.

OpenGL

`NSOpenGLGetOption` (page 44)
Returns global OpenGL options.

`NSOpenGLGetVersion` (page 45)
Returns the NSOpenGL version numbers.

`NSOpenGLSetOption` (page 45)
Sets global OpenGL options.

Panels

`NSBeginAlertSheet` (page 18)
Creates and runs an alert sheet.

`NSBeginCriticalAlertSheet` (page 20)
Creates and runs a critical alert sheet.

`NSBeginInformationalAlertSheet` (page 20)
Creates and runs an informational alert sheet.

`NSGetAlertPanel` (page 39)
Returns an alert panel.

`NSGetCriticalAlertPanel` (page 40)
Returns an alert panel to display a critical message.

`NSGetInformationalAlertPanel` (page 41)
Returns an alert panel to display an informational message.

`NSReleaseAlertPanel` (page 53)
Releases an alert panel.

`NSRunAlertPanel` (page 53)
Creates an alert panel.

`NSRunCriticalAlertPanel` (page 54)
Creates and runs a critical alert panel.

`NSRunInformationalAlertPanel` (page 55)
Creates and runs an informational alert panel.

Pasteboards

`NSCreateFileContentsPboardType` (page 24)
Returns a pasteboard type based on the passed file type.

`NSCreateFilenamePboardType` (page 25)
Returns a pasteboard type based on the passed file type.

`NSGetFileType` (page 40)
Returns a file type based on the passed pasteboard type.

`NSGetFileTypes` (page 40)
Returns an array of file types based on the passed pasteboard types.

System Beep

Additional information on sounds can be found in `NSSound`.

`NSBeep` (page 18)
Plays the system beep.

Functions

NSAccessibilityActionDescription

Returns a standard description for an action.

```
NSString * NSAccessibilityActionDescription (
   NSString *action
);
```

Discussion
This function returns a standard description for *action*.

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

Related Sample Code
Dicey
ImageMapExample

Declared In
`NSAccessibility.h`

NSAccessibilityPostNotification

Sends a notification to any observing assistive applications.

```
void NSAccessibilityPostNotification (
   id element,
   NSString *notification
);
```

Discussion
Sends *notification* to any assistive applications that have registered to receive the notification from the user interface object *element* in your application. Accessibility notifications require special handling, so they cannot be posted using `NSNotificationCenter`.

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

Related Sample Code
Dicey
TrackBall

Declared In
`NSAccessibility.h`

NSAccessibilityRaiseBadArgumentException

Raises an error if the parameter is the wrong type or has an illegal value

```
void NSAccessibilityRaiseBadArgumentException (
   id element,
   NSString *attribute,
   id value
);
```

Discussion
Raises an error if a parameter is the wrong type or has an illegal value. This function can also be used to raise an error if an attempt is made to set an attribute's value with the wrong type or an illegal value.

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

Declared In
`NSAccessibility.h`

NSAccessibilityRoleDescription

Returns a standard description for a role and subrole.

```
NSString * NSAccessibilityRoleDescription (
   NSString *role,
   NSString *subrole
);
```

Discussion
You should pass nil to this function if there is no subrole. This function returns a description of a standard role. For example, if you implement a button widget that does not inherit from `NSButton`, you should use this function to return a localized role description matching that returned by a standard button.

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

Related Sample Code
Dicey
ImageMapExample
TrackBall

Declared In
`NSAccessibility.h`

NSAccessibilityRoleDescriptionForUIElement

Returns a standard role description for a user interface element.

```
NSString * NSAccessibilityRoleDescriptionForUIElement (
   id element
);
```

Discussion
This function is like the `NSAccessibilityRoleDescription` (page 13) function, except that it queries *`element`* to get the role and subrole. The `NSAccessibilityRoleDescription` function is more efficient, but this function is useful for accessorizing base classes so that they properly handle derived classes, which may override the subrole or even the role.

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

Related Sample Code
ImageMapExample

Declared In
`NSAccessibility.h`

NSAccessibilityUnignoredAncestor

Returns an unignored accessibility object, ascending the hierarchy if necessary.

```
id NSAccessibilityUnignoredAncestor (
    id element
);
```

Discussion
Tests whether *element* is an ignored object, returning either *element*, if it is not ignored, or the first unignored ancestor of *element*.

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

Related Sample Code
Dicey
ImageMapExample
TrackBall

Declared In
NSAccessibility.h

NSAccessibilityUnignoredChildren

Returns a list of unignored accessibility objects, descending the hierarchy if necessary.

```
NSArray * NSAccessibilityUnignoredChildren (
   NSArray *originalChildren
);
```

Discussion
Returns a copy of *originalChildren* with any ignored objects in the array replaced by their unignored descendants.

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

Related Sample Code
Dicey
ImageMapExample

Declared In
NSAccessibility.h

NSAccessibilityUnignoredChildrenForOnlyChild

Returns a list of unignored accessibility objects, descending the hierarchy if necessary.

```
NSArray * NSAccessibilityUnignoredChildrenForOnlyChild (
   id originalChild
);
```

Discussion
Tests whether *originalChild* is an ignored object and returns an array containing either *originalChild*, if it is not ignored, or its unignored descendants.

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

Declared In
`NSAccessibility.h`

NSAccessibilityUnignoredDescendant

Returns an unignored accessibility object, descending the hierarchy if necessary.

```
id NSAccessibilityUnignoredDescendant (
   id element
);
```

Discussion
Tests whether *`element`* is an ignored object, returning either *`element`*, if it is not ignored, or the first unignored descendant of *`element`*. Use this function only if you know there is a linear, one-to-one, hierarchy below *`element`*. Otherwise, if *`element`* has either no unignored children or multiple unignored children, this function fails and returns `nil`.

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

Related Sample Code
ImageMapExample

Declared In
`NSAccessibility.h`

NSApplicationLoad

Startup function to call when running Cocoa code from a Carbon application.

```
BOOL NSApplicationLoad (void);
```

Return Value
`YES` if the `NSApplication` object was successfully initialized and can now be used from your Carbon application or `NO` if there was an error during initialization.

Discussion
You typically call this function before calling other Cocoa code in a plug-in loaded into a primarily Carbon application. If the shared `NSApplication` object is not already initialized, this function initializes it and sets up the necessary event handlers for Cocoa.

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

Related Sample Code
CarbonCocoaCoreImageTab
CarbonQuartzComposer_TV
CocoaInCarbon
CrossEvents
HIView-NSView

Declared In
NSApplication.h

NSApplicationMain

Called by the main function to create and run the application.

```
int NSApplicationMain (
   int argc,
   const char *argv[]
);
```

Parameters

argc

The number of arguments in the argv parameter.

argv

An array of pointers containing the arguments that were passed to the application at startup.

Return Value
This method never returns a result code. Instead, it calls the `exit` function to exit the application and terminate the process. If you want to determine why the application exited, you should look at the result code from the `exit` function instead.

Discussion
Creates the application, loads the main nib file from the application's main bundle, and runs the application. You must call this function from the main thread of your application, and you typically call it only once from your application's `main` function, which is usually generated automatically by Xcode.

Special Considerations

`NSApplicationMain` itself ignores the *argc* and *argv* arguments. Instead, Cocoa gets its arguments indirectly via `_NSGetArgv`, `_NSGetArgc`, and `_NSGetEnviron` (see <crt_externs.h>).

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

Related Sample Code
CoreRecipes
Dicey
ImageClient
MyPhoto
Quartz EB

Declared In
NSApplication.h

NSAvailableWindowDepths

Returns the available window depth values.

```
const NSWindowDepth * NSAvailableWindowDepths (void);
```

Discussion
Returns a null-terminated array of `NSWindowDepthâ  Window Depth` values that specify which window depths are currently available. Window depth values are converted to specific display properties using the functions `NSBitsPerPixelFromDepth` (page 22), `NSBitsPerSampleFromDepth` (page 22), `NSColorSpaceFromDepth` (page 22), and `NSPlanarFromDepth` (page 46).

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

Declared In
`NSGraphics.h`

NSBeep

Plays the system beep.

```
void NSBeep (void);
```

Discussion
Plays the system beep. Users can select a sound to be played as the system beep. On a Macintosh computer, for example, you can change sounds with the Sound pane of System Preferences.

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

Related Sample Code
GridCalendar
NewsReader
PDFKitLinker2
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSGraphics.h`

NSBeginAlertSheet

Creates and runs an alert sheet.

```
void NSBeginAlertSheet (
   NSString *title,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   NSWindow *docWindow,
   id modalDelegate,
   SEL didEndSelector,
   SEL didDismissSelector,
   void *contextInfo,
   NSString *msg,
   ...
);
```

Discussion
Creates and runs an alert sheet on *docWindow*, with the title of *title*, the text of *msg*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*.

The buttons are laid out on the lower-right corner of the sheet, with *defaultButton* on the right, *alternateButton* on the left, and *otherButton* in the middle. If *title* is `nil` or an empty string, a default localized title is used ("Alert" in English). If *defaultButton* is `nil` or an empty string, a default localized button title ("OK" in English) is used. For the remaining buttons, this function creates them only if their corresponding button title is non-`nil`.

A Command-D key equivalent for the "Don't Save" button is provided, if one is found. The button titles are searched for the localized value for "Don't Save." If a match is found, that button is assigned a Command-D key equivalent, provided it is not the default button.

If you create a modal panel using `runModalForWindow:` or `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:`, you can assign the key equivalent yourself, using `setKeyEquivalent:` and `setKeyEquivalentModifierMask:`.

The *msg* argument is the message that's displayed in the panel. It can use printf-style formatting characters; any necessary arguments should be listed at the end of the function's argument list (after the *msg* argument). For more information on formatting characters, see the man page for `printf`.

When the modal session is ended, and before the sheet is dismissed, the *didEndSelector* is invoked on the *modalDelegate*. passing *contextInfo*. After the sheet is dismissed, the *didDismissSelector* is invoked on the *modalDelegate*, passing *contextInfo*. Typically, you will want to implement the *didEndSelector* but you may pass `NULL` for the *didDismissSelector*. The two selectors should be defined as follows:

```
sheetDidEnd:(NSWindow *)sheet returnCode:(int)returnCode contextInfo:(void
*)contextInfo;
sheetDidDismiss:(NSWindow *)sheet returnCode:(int)returnCode contextInfo:(void
  *)contextInfo;
```

where *sheet* is the alert sheet, *returnCode* specifies which button the user pressed, and *contextInfo* is the same *contextInfo* passed into `NSBeginAlertSheet`. *returnCode* can be one of the following:

- `NSAlertDefaultReturn` means the user pressed the default button.
- `NSAlertAlternateReturn` means the user pressed the alternate button.
- `NSAlertOtherReturn` means the user pressed the other button.
- `NSAlertErrorReturn` means an error occurred while running the alert panel.

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

Related Sample Code
ImageClient
PrefsPane
QTAudioExtractionPanel
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSPanel.h`

NSBeginCriticalAlertSheet

Creates and runs a critical alert sheet.

```
void NSBeginCriticalAlertSheet (
   NSString *title,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   NSWindow *docWindow,
   id modalDelegate,
   SEL didEndSelector,
   SEL didDismissSelector,
   void *contextInfo,
   NSString *msg,
   ...
);
```

Discussion
Creates and runs a critical alert sheet on *docWindow*, with the title of *title*, the text of *msg*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*.

See the description of `NSBeginAlertSheet` (page 18) for information on layout, default parameters, and the selectors.

The sheet presented to the user is badged with a caution icon. Critical alerts should be used only as specified in the "Alerts" section of the Windows chapter of *Apple Human Interface Guidelines*.

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

Declared In
`NSPanel.h`

NSBeginInformationalAlertSheet

Creates and runs an informational alert sheet.

```
void NSBeginInformationalAlertSheet (
   NSString *title,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   NSWindow *docWindow,
   id modalDelegate,
   SEL didEndSelector,
   SEL didDismissSelector,
   void *contextInfo,
   NSString *msg,
   ...
);
```

Discussion
Creates and runs an informational alert sheet on *docWindow*, with the title of *title*, the text of *msg*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*.

See the description of `NSBeginAlertSheet` (page 18) for information on layout, default parameters, and the selectors.

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

Related Sample Code
AutoSample
InstallerPluginSample

Declared In
`NSPanel.h`

NSBestDepth

Attempts to return a window depth adequate for the specified parameters.

```
NSWindowDepth NSBestDepth (
   NSString *colorSpace,
   NSInteger bps,
   NSInteger bpp,
   BOOL planar,
   BOOL *exactMatch
);
```

Discussion
Returns a window depth deep enough for the given number of colors in *colorSpace*, bits per sample specified by *bps*, bits per pixel specified by *bpp*, and whether planar as specified by *planar*. Upon return, the variable pointed to by *exactMatch* is `YES` if the window depth can accommodate all of the values specified by the parameters, `NO` if it can't.

Use this function to compute window depths. This function tries to accommodate all the parameters (match or better); if there are multiple matches, it gives the closest, with matching *colorSpace* first, then *bps*, then *planar*, then *bpp*. *bpp* is "bits per pixel"; 0 indicates default (same as the number of bits per plane, either *bps* or `bps` * `NSNumberOfColorComponents` (page 44)); other values may be used as hints to provide backing stores of different configuration; for instance, 8-bit color. The *exactMatch* parameter is optional and indicates whether all the parameters matched exactly.

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

Declared In
`NSGraphics.h`

NSBitsPerPixelFromDepth

Returns the bits per pixel for the specified window depth.

```
NSInteger NSBitsPerPixelFromDepth (
   NSWindowDepth depth
);
```

Discussion
Returns the number of bits per pixel for the window depth specified by *depth*.

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

Declared In
`NSGraphics.h`

NSBitsPerSampleFromDepth

Returns the bits per sample for the specified window depth.

```
NSInteger NSBitsPerSampleFromDepth (
   NSWindowDepth depth
);
```

Discussion
Returns the number of bits per sample (bits per pixel in each color component) for the window depth specified by *depth*.

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

Declared In
`NSGraphics.h`

NSColorSpaceFromDepth

Returns the name of the color space corresponding to the passed window depth.

```
NSString * NSColorSpaceFromDepth (
   NSWindowDepth depth
);
```

Discussion
Returns the color space name for the specified *depth*. For example, the returned color space name can be `NSCalibratedRGBColorSpace`, or `NSDeviceCMYKColorSpace`.

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

Declared In
`NSGraphics.h`

NSConvertGlyphsToPackedGlyphs

Prepares a set of glyphs for processing by character-based routines.

```
NSInteger NSConvertGlyphsToPackedGlyphs (
   NSGlyph *glBuf,
   NSInteger count,
   NSMultibyteGlyphPacking packing,
   char *packedGlyphs
);
```

Discussion
Takes a buffer of glyphs, specified in the *glBuf* parameter, and packs them into a condensed character array. The character array is returned in the *packedGlyphs* parameter, which should have enough space for at least ((4 * count) + 1) bytes to guarantee that the packed glyphs fit. *count* specifies the number of glyphs in *glBuf*. *packing* specifies how the glyphs are currently packed.

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

Declared In
`NSFont.h`

NSCopyBits

Copies a bitmap image to the location specified by a destination point.

```
void NSCopyBits (
   NSInteger srcGState,
   NSRect srcRect,
   NSPoint destPoint
);
```

Discussion
Copies the pixels in the rectangle specified by *srcRect* to the location specified by *destPoint*. The source rectangle is defined in the graphics state designated by *srcGState*. If *srcGState* is `NSNullObject`, the current graphics state is assumed. The *destPoint* destination is defined in the current graphics state.

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

Declared In
`NSGraphics.h`

NSCountWindows

Counts the number of onscreen windows.

```
void NSCountWindows (
   NSInteger *count
);
```

Parameters

count

On output, this parameter contains the number of onscreen windows.

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

Declared In
NSGraphics.h

NSCountWindowsForContext

Counts the number of onscreen windows belonging to a particular application.

```
void NSCountWindowsForContext (
   NSInteger context,
   NSInteger *count
);
```

Discussion
Counts the number of onscreen windows belonging to a particular application, identified by *context*, which is a window server connection ID. The function returns the number by reference in *count*.

Use of this function is discouraged as it may be deprecated in a future release.

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

Declared In
NSGraphics.h

NSCreateFileContentsPboardType

Returns a pasteboard type based on the passed file type.

```
NSString * NSCreateFileContentsPboardType (
   NSString *fileType
);
```

Discussion
Returns an NSString to a pasteboard type representing a file's contents based on the supplied string *fileType*. *fileType* should generally be the extension part of a filename. The conversion from a named file type to a pasteboard type is simple; no mapping to standard pasteboard types is attempted.

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

Declared In
NSPasteboard.h

NSCreateFilenamePboardType

Returns a pasteboard type based on the passed file type.

```
NSString * NSCreateFilenamePboardType (
   NSString *fileType
);
```

Discussion
Returns an `NSString` to a pasteboard type representing a filename based on the supplied string *`fileType`*.

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

Declared In
`NSPasteboard.h`

NSDisableScreenUpdates

Disables screen updates.

```
void NSDisableScreenUpdates (void);
```

Discussion
Prevents drawing operations from being flushed to the screen for all windows belonging to the calling process. When you re-enable screen updates (with `NSEnableScreenUpdates` (page 36)) screen flushing for all windows belonging to the calling process appears to be simultaneous. You typically call this function so that operations on multiple windows appear atomic to the user. This is a technique particularly useful for synchronizing parent and child windows. Make sure that the period after calling this function and before reenabling updates is short; the system only allow updating to be disabled for a limited time (currently one second) before automatically reenabling updates. Successive calls to this function are placed on a stack and must be popped off that stack by matching `NSEnableScreenUpdates` calls.

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

Declared In
`NSGraphics.h`

NSDottedFrameRect

Draws a bordered rectangle.

```
void NSDottedFrameRect (
   NSRect aRect
);
```

Discussion
Deprecated. Use a dashed `NSBezierPath` instead.

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

See Also
`NSDrawTiledRects` (page 33)

Declared In
NSGraphics.h

NSDrawBitmap

Draws a bitmap image.

```
void NSDrawBitmap (
   NSRect rect,
   NSInteger width,
   NSInteger height,
   NSInteger bps,
   NSInteger spp,
   NSInteger bpp,
   NSInteger bpr,
   BOOL isPlanar,
   BOOL hasAlpha,
   NSString *colorSpaceName,
   const unsigned char *const data[5]
);
```

Discussion
This function is marginally obsolete. Most applications are better served using the `NSBitmapImageRep` class to read and display bitmap images.

This function renders an image from a bitmap, binary data that describes the pixel values for the image.

This function renders a bitmap image using an appropriate display operator. It puts the image in the rectangular area specified by its first argument, *`rect`*; the rectangle is specified in the current coordinate system and is located in the current window. The next two arguments, *`pixelsWide`* and *`pixelsHigh`*, give the width and height of the image in pixels. If either of these dimensions is larger or smaller than the corresponding dimension of the destination rectangle, the image will be scaled to fit.

The remaining arguments describe the bitmap data, as explained in the following paragraphs.

The *`bitsPerSample`* argument is the number of bits per sample for each pixel and *`samplesPerPixel`* is the number of samples per pixel. *`bitsPerPixel`* is based on *`samplesPerPixel`* and the configuration of the bitmap: if the configuration is planar, then the value of *`bitsPerPixel`* should equal the value of *`bitsPerSample`*; if the configuration isn't planar (is meshed instead), *`bitsPerPixel`* should equal `bitsPerSample * samplesPerPixel`.

The *`bytesPerRow`* argument is calculated in one of two ways, depending on the configuration of the image data (data configuration is described below). If the data is planar, `bytesPerRow is (7 + (pixelsWide * bitsPerSample)) / 8`. If the data is meshed, `bytesPerRow is (7 + (pixelsWide * bitsPerSample * samplesPerPixel)) / 8`.

A sample is data that describes one component of a pixel. In an RGB color system, the red, green, and blue components of a color are specified as separate samples, as are the cyan, magenta, yellow, and black components in a CMYK system. Color values in a grayscale are a single sample. Alpha values that determine transparency and opaqueness are specified as a coverage sample separate from color. In bitmap images with alpha, the color (or gray) components have to be premultiplied with the alpha. This is the way images with alpha are displayed, this is the way they are read back, and this is the way they are stored in TIFFs.

The *isPlanar* argument refers to the way data is configured in the bitmap. This flag should be set to YES if a separate data channel is used for each sample. The function provides for up to five channels, *data1*, *data2*, *data3*, *data4*, and *data5*. It should be set NO if sample values are interwoven in a single channel (meshed); all values for one pixel are specified before values for the next pixel.

Grayscale windows store pixel data in planar configuration; color windows store it in meshed configuration. NSDrawBitmap can render meshed data in a planar window, or planar data in a meshed window. However, it's more efficient if the image has a depth (*bitsPerSample*) and configuration (*isPlanar*) that match the window.

The *hasAlpha* argument indicates whether the image contains alpha. If it does, the number of samples should be 1 greater than the number of color components in the model (for example, 4 for RGB).

The *colorSpace* argument can be NSCustomColorSpace, indicating that the image data is to be interpreted according to the current color space in the graphics state. This allows for imaging using custom color spaces. The image parameters supplied as the other arguments should match what the color space is expecting.

If the image data is planar, *data*[0] through *data*[*samplesPerPixel*–1] point to the planes; if the *data* is meshed, only *data*[0] needs to be set.

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

Declared In
NSGraphics.h

NSDrawButton

Draws a gray-filled rectangle representing a user-interface button.

```
void NSDrawButton (
   NSRect aRect,
   NSRect clipRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw. Only those parts of *aRect* that lie within the *clipRect* are actually drawn.

clipRect

The clipping rectangle to use during drawing.

Discussion
Draws a gray-filled rectangle, used to signify a user-interface button. Since this function is often used to draw the border of a view, the *aRect* parameter typically contains the view's bounds rectangle. For an Aqua button, use an NSButton object instead.

This function fills the specified rectangle with light gray. This function is designed for rectangles that are defined in unscaled, unrotated coordinate systems (that is, where the y axis is vertical, the x axis is horizontal, and a unit along either axis is equal to 1 screen pixel). The coordinate system can be either flipped or unflipped. The sides of the rectangle should lie on pixel boundaries.

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

Declared In
NSGraphics.h

NSDrawColorTiledRects

Draws a colored bordered rectangle.

```
NSRect NSDrawColorTiledRects (
   NSRect boundsRect,
   NSRect clipRect,
   const NSRectEdge *sides,
   NSColor **colors,
   NSInteger count
);
```

Parameters

boundsRect
The bounding rectangle (in the current coordinate system) in which to draw. Since this function is often used to draw the border of a view, this rectangle will typically be that view's bounds rectangle. Only those parts of *boundsRect* that lie within the *clipRect* are actually drawn.

clipRect
The clipping rectangle to use during drawing.

sides
The sides of the rectangle for which you want to specify custom colors. Each side must have a corresponding entry in the *colors* parameter.

colors
The colors to draw for each of the edges listed in the *sides* parameter.

count
The number of 1.0-unit-wide slices to draw on the specified sides.

Return Value
The rectangle that lies within the resulting border.

Discussion
Behaves the same as NSDrawTiledRects (page 33) except it draws its border using colors from the *colors* array.

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

Declared In
NSGraphics.h

NSDrawDarkBezel

Draws a dark gray-filled rectangle with a bezel border.

```
void NSDrawDarkBezel (
   NSRect aRect,
   NSRect clipRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw. Only those parts of *aRect* that lie within the *clipRect* are actually drawn.

clipRect

The clipping rectangle to use during drawing.

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

See Also
NSDrawTiledRects (page 33)

Related Sample Code
BindingsJoystick

Declared In
NSGraphics.h

NSDrawGrayBezel

Draws a gray-filled rectangle with a bezel border.

```
void NSDrawGrayBezel (
   NSRect aRect,
   NSRect clipRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw. Only those parts of *aRect* that lie within the *clipRect* are actually drawn.

clipRect

The clipping rectangle to use during drawing.

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

See Also
NSDrawTiledRects (page 33)

Declared In
NSGraphics.h

NSDrawGroove

Draws a gray-filled rectangle with a groove border.

```
void NSDrawGroove (
   NSRect aRect,
   NSRect clipRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw. Only those parts of *aRect* that lie within the *clipRect* are actually drawn.

clipRect

The clipping rectangle to use during drawing.

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

See Also
NSDrawTiledRects (page 33)

Declared In
NSGraphics.h

NSDrawLightBezel

Draws a white-filled rectangle with a bezel border.

```
void NSDrawLightBezel (
   NSRect aRect,
   NSRect clipRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw. Only those parts of *aRect* that lie within the *clipRect* are actually drawn.

clipRect

The clipping rectangle to use during drawing.

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

See Also
NSDrawTiledRects (page 33)

Related Sample Code
BindingsJoystick

Declared In
NSGraphics.h

NSDrawNinePartImage

Draws a nine-part tiled image.

```
void NSDrawNinePartImage(NSRect frame,
   NSImage *topLeftCorner,
   NSImage *topEdgeFill,
   NSImage *topRightCorner,
   NSImage *leftEdgeFill,
   NSImage *centerFill,
   NSImage *rightEdgeFill,
   NSImage *bottomLeftCorner,
   NSImage *bottomEdgeFill,
   NSImage *bottomRightCorner,
   NSCompositingOperation op,
   CGFloat alphaFraction,
   BOOL flipped
);
```

Parameters

frame

The rectangle (specified in the current coordinate system) in which to draw the images.

topLeftCorner

The image to display in the top-left corner.

topEdgeFill

The image used to tile the space between the *topLeftCorner* and *topRightCorner* images.

topRightCorner

The image to display in the top-right corner.

leftEdgeFill

The image used to tile the space between the *topLeftCorner* and *bottomLeftCorner* images.

centerFill

The image used to tile the center area between the other eight images.

rightEdgeFill

The image used to tile the space between the *topRightCorner* and *bottomRightCorner* images.

bottomLeftCorner

The image to display in the bottom-left corner.

bottomEdgeFill

The image used to tile the space between the *bottomLeftCorner* and *bottomRightCorner* images.

bottomRightCorner

The image to display in the bottom-right corner.

op

The compositing operation to use when rendering the images.

alphaFraction

The alpha value to apply to the rendered image. This value can range between 0.0 and 1.0, with 0.0 being fully transparent and 1.0 being fully opaque.

flipped

Specify `YES` if you are drawing the images in a flipped coordinate system; otherwise, specify `NO`.

Discussion

This function is typically used to draw custom cells that are capable of being resized both vertically and horizontally. Cells of this type are comprised of four fixed-size corner images along and a set of edge and center images that are used to fill the gaps between the corners. These cells allow you to create sophisticated looking controls that can grow and shrink in any direction without distorting the control's overall appearance.

You should prefer the use of this function over your own custom code for handling multi-part images whose size can change. This function correctly manages the subtle behaviors needed to handle resolution independence issues and to avoid visual artifacts caused by tiling the various images.

This function uses the top-left and bottom-right corner images to determine the widths and heights of the edge areas that need to be filled. If the width or height of the bottom-left and top-right images are not sized appropriately, they may be scaled to fill their corner area. Edge areas between the corners are tiled using the corresponding image. Similarly, the center area is tiled using the specified center image.

The *flipped* parameter lets you reorient the contents of each image when drawing in a flipped coordinate system. By default, images use an internal coordinate system that is not flipped. Rendering such an image in a flipped coordinate system would therefore cause the image to appear upside down. Passing `YES` for the *flipped* parameter adjusts the image's internal coordinate system to draw it correctly in a flipped environment.

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

Declared In
`NSCell.h`

NSDrawThreePartImage

Draws a three-part tiled image.

```
void NSDrawThreePartImage(NSRect frame,
   NSImage *startCap,
   NSImage *centerFill,
   NSImage *endCap,
   BOOL vertical,
   NSCompositingOperation op,
   CGFloat alphaFraction,
   BOOL flipped
);
```

Parameters

frame
: The rectangle (specified in the current coordinate system) in which to draw the images.

startCap
: For a horizontal three-part image, this is the image located at the left edge of the frame rectangle. For a vertical three-part image, this image appears at the top of the screen in an unflipped coordinate system and at the bottom of the screen in a flipped coordinate system.

centerFill
: The image used to tile the space between the *startCap* and *endCap* images.

endCap
: For a horizontal three-part image, this is the image located at the right edge of the frame rectangle. For a vertical three-part image, this image appears at the bottom of the screen in an unflipped coordinate system and at the top of the screen in a flipped coordinate system.

vertical
: Specify `YES` if the images should be stacked on top of one another to create a vertically oriented element. Specify `NO` if the images should be laid out side-by-side to create a horizontally oriented element.

op

The compositing operation to use when rendering the images.

alphaFraction

The alpha value to apply to the rendered image. This value can range between 0.0 and 1.0, with 0.0 being fully transparent and 1.0 being fully opaque.

flipped

Specify `YES` if you are drawing the images in a flipped coordinate system; otherwise, specify `NO`.

Discussion
This function is typically used to draw custom cells (such as the backgrounds for push button and slider controls) that are capable of being resized along a single axis only. Cells of this type are comprised of fixed-size end cap images and a center area that is filled by tiling the specified center image as many times as needed to fill the gap. These cells allow you to create sophisticated looking controls that can grow and shrink without distorting the control's overall appearance.

You should prefer the use of this function over your own custom code for handling multi-part images whose size can change. This function correctly manages the subtle behaviors needed to handle resolution independence issues and to avoid visual artifacts caused by tiling the various images.

When drawing a horizontally oriented control, the images in the *startCap*, *centerFill*, and *endCap* parameters should all have the same height, and that height should match the height of the frame rectangle. If an image's height does not match the height of the frame rectangle, it is scaled until it does match, which might yield less desirable results. For vertically oriented controls, the image widths are scaled instead of the heights.

The *flipped* parameter lets you reorient the contents of each image when drawing in a flipped coordinate system. By default, images use an internal coordinate system that is not flipped. Rendering such an image in a flipped coordinate system would therefore cause the image to appear upside down. Passing `YES` for the *flipped* parameter adjusts the image's internal coordinate system to draw it correctly in a flipped environment.

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

Declared In
`NSCell.h`

NSDrawTiledRects

Draws rectangles with borders.

```
NSRect NSDrawTiledRects (
   NSRect boundsRect,
   NSRect clipRect,
   const NSRectEdge *sides,
   const CGFloat *grays,
   NSInteger count
);
```

Parameters

boundsRect

The bounding rectangle (in the current coordinate system) in which to draw. Since this function is often used to draw the border of a view, this rectangle will typically be that view's bounds rectangle. Only those parts of *boundsRect* that lie within the *clipRect* are actually drawn.

clipRect

The clipping rectangle to use during drawing.

sides

The sides of the rectangle for which you want to specify custom gray levels. Each side must have a corresponding entry in the *grays* parameter.

grays

The gray levels to draw for each of the edges listed in the *sides* parameter.

count

The number of 1.0-unit-wide slices to draw on the specified sides.

Return Value

The rectangle that lies within the resulting border.

Discussion

This is a generic function that can be used to draw different types of borders inside a given rectangle. These borders can be used to outline an area or to give rectangles the effect of being recessed from or elevated above the surface of the screen.

The *sides*, *grays*, and *count* parameters determine how thick the border is and what gray levels are used to form it. This function uses the `NSDivideRect` function to take successive 1.0-unit-wide slices from the sides of the rectangle specified by the *sides* parameter. Each slice is drawn using the corresponding gray level from the *grays* parameter. This function makes and draws these slices *count* number of times. If you specify the same side more than once, the second slice is drawn inside the first.

The following example uses this function to draw a bezeled border consisting of a 1.0–unit-wide white line at the top and on the left side and a 1.0-unit-wide dark-gray line inside a 1.0–unit-wide black line on the other two sides. The resulting rectangle inside this border is then filled in using light gray.

```
NSRectEdge mySides[] = {NSMinYEdge, NSMaxXEdge, NSMaxYEdge, NSMinXEdge,
                        NSMinYEdge, NSMaxXEdge};
float myGrays[] = {NSBlack, NSBlack, NSWhite, NSWhite,
                        NSDarkGray, NSDarkGray};
NSRect aRect, clipRect; // Assume exists

aRect = NSDrawTiledRects(aRect, clipRect, mySides, myGrays, 6);
[[NSColor grayColor] set];
NSRectFill(aRect);
```

In the preceding example, *mySides* is an array that specifies sides of a rectangle; for example, `NSMinYEdge` selects the side parallel to the x axis with the smallest y coordinate value. *myGrays* is an array that specifies the successive gray levels to be used in drawing parts of the border.

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

Declared In
NSGraphics.h

NSDrawWhiteBezel

Draws a white-filled rectangle with a bezel border.

```
void NSDrawWhiteBezel (
   NSRect aRect,
   NSRect clipRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw. Only those parts of *aRect* that lie within the *clipRect* are actually drawn.

clipRect

The clipping rectangle to use during drawing.

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

See Also
NSDrawTiledRects (page 33)

Related Sample Code
Sketch-112

Declared In
NSGraphics.h

NSDrawWindowBackground

Draws the window's default background pattern into the specified rectangle of the currently focused view.

```
void NSDrawWindowBackground (
   NSRect aRect
);
```

Parameters

aRect

The rectangle (in the current coordinate system) in which to draw the window's background pattern.

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

Declared In
NSGraphics.h

NSEnableScreenUpdates

Enables screen updates

```
void NSEnableScreenUpdates (void);
```

Discussion
Reenables, for all windows of a process, the flushing of drawing operations to the screen that was previously disabled by `NSDisableScreenUpdates` (page 25). Successive calls to `NSDisableScreenUpdates` are placed on a stack and must be popped off that stack by matching calls to this function.

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

Declared In
`NSGraphics.h`

NSEraseRect

Erases the specified rect by filling it with white.

```
void NSEraseRect (
   NSRect aRect
);
```

Parameters

aRect
The rectangle (in the current coordinate system) defining the area to erase.

Discussion
This function fills the specified rectangle with white. It does not alter the current color.

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

Related Sample Code
Quartz EB

Declared In
`NSGraphics.h`

NSEventMaskFromType

Returns the event mask for the specified type.

```
static NSUInteger NSEventMaskFromType (
   NSEventType type
);
```

Parameters

type
The event type whose mask you want to get.

Return Value
The event mask corresponding to the specified type. The returned mask is equivalent to the number 1 left-shifted by *type* bits.

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

Related Sample Code
Quartz Composer Matrix

Declared In
NSEvent.h

NSFrameRect

Draw a bordered rectangle.

```
void NSFrameRect (
   NSRect aRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw.

Discussion
Draws a frame around the inside of *aRect* in the current color and using the NSCompositeCopy compositing operation. The width is equal to 1.0 in the current coordinate system. Since the frame is drawn inside the rectangle, it will be visible even if drawing is clipped to the rectangle.

Because this function does not draw directly on the line, but rather inside it, it uses the current fill color (not stroke color) when drawing.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

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

See Also
NSDrawTiledRects (page 33)

Related Sample Code
Cropped Image
Dicey
FilterDemo
PDF Calendar
Sketch-112

Declared In
NSGraphics.h

NSFrameRectWithWidth

Draw a bordered rectangle.

```
void NSFrameRectWithWidth (
   NSRect aRect,
   CGFloat frameWidth
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw.

frameWidth

The width of the frame, specified in points.

Discussion

Draws a frame around the inside of *aRect* in the current color and using the `NSCompositeCopy` compositing operation. The width is equal to *frameWidth* in the current coordinate system. Since the frame is drawn inside the rectangle, it will be visible even if drawing is clipped to the rectangle.

Because this function does not draw directly on the line, but rather inside it, it uses the current fill color (not stroke color) when drawing.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSDrawTiledRects` (page 33)

Related Sample Code

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSGraphics.h`

NSFrameRectWithWidthUsingOperation

Draw a bordered rectangle using the specified compositing operation.

```
void NSFrameRectWithWidthUsingOperation (
   NSRect aRect,
   CGFloat frameWidth,
   NSCompositingOperation op
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw.

frameWidth

The width of the frame, specified in points.

op

The compositing operation to use when drawing the frame.

Discussion
Draws a frame around the inside of *aRect* in the current color, using the compositing operation *op*. The width is equal to *frameWidth* in the current coordinate system. Since the frame is drawn inside the rectangle, it will be visible even if drawing is clipped to the rectangle.

Because this function does not draw directly on the line, but rather inside it, it uses the current fill color (not stroke color) when drawing.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

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

Related Sample Code
EnhancedAudioBurn
PDF Annotation Editor

Declared In
NSGraphics.h

NSGetAlertPanel

Returns an alert panel.

```
id NSGetAlertPanel (
   NSString *title,
   NSString *msg,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   ...
);
```

Discussion
Returns an `NSPanel` that can be used to set up a modal session. A modal session is useful for allowing the user to interrupt the program. During a modal session, you can perform activities while the panel is displayed and check at various points in your program whether the user has clicked one of the panel's buttons. The arguments for this function are the same as those for `NSRunAlertPanel` (page 53), but unlike that function, no button is displayed if *defaultButton* is `nil`.

To set up a modal session, send the Application object `beginModalSessionForWindow:` with the panel returned by `NSGetAlertPanel` as its argument. When you want to check if the user has clicked one of the panel's buttons, use `runModalSession:`. To end the modal session, use `endModalSession:`. When you're finished with the panel created by `NSGetAlertPanel`, you must free it by passing it to `NSReleaseAlertPanel` (page 53).

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

Declared In
NSPanel.h

NSGetCriticalAlertPanel

Returns an alert panel to display a critical message.

```
id NSGetCriticalAlertPanel (
   NSString *title,
   NSString *msg,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   ...
);
```

Discussion
Returns an `NSPanel` that can be used to set up a modal session. No button is displayed if *`defaultButton`* is `nil`. When you're finished with the panel created by this function, you must free it by passing it to `NSReleaseAlertPanel` (page 53).

The arguments for this function are the same as those for the `NSGetAlertPanel` (page 39). For more information on using a panel in a modal session, see `NSGetAlertPanel`.

The panel presented to the user is badged with a caution icon. Critical alerts should be used only as specified in the "Alerts" section of the Windows chapter of *Apple Human Interface Guidelines*.

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

Declared In
`NSPanel.h`

NSGetFileType

Returns a file type based on the passed pasteboard type.

```
NSString * NSGetFileType (
   NSString *pboardType
);
```

Discussion
This function is the inverse of both `NSCreateFileContentsPboardType` (page 24) and `NSCreateFilenamePboardType` (page 25). When passed a pasteboard type as returned by those functions, it returns the extension or filename from which the type was derived. It returns `nil` if *`pboardType`* isn't a pasteboard type created by those functions.

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

Declared In
`NSPasteboard.h`

NSGetFileTypes

Returns an array of file types based on the passed pasteboard types.

```
NSArray * NSGetFileTypes (
   NSArray *pboardTypes
);
```

Discussion
Accepts a null-terminated array of pointers to pasteboard types and returns a null-terminated array of the unique extensions and filenames from the file content and filename types found in the input array. It returns `nil` if the input array contains no file content or filename types. The returned array is allocated and must be freed by the caller. The pointers in the return array point into strings passed in the input array.

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

Declared In
`NSPasteboard.h`

NSGetInformationalAlertPanel

Returns an alert panel to display an informational message.

```
id NSGetInformationalAlertPanel (
   NSString *title,
   NSString *msg,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   ...
);
```

Discussion
Returns an NSPanel that can be used to set up a modal session. No button is displayed if *`defaultButton`* is `nil`. When you're finished with the panel created by this function, you must free it by passing it to `NSReleaseAlertPanel` (page 53).

The arguments for this function are the same as those for the `NSRunAlertPanel` (page 53) function. For more information on using a panel in a modal session, see `NSGetAlertPanel` (page 39).

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

Declared In
`NSPanel.h`

NSGetWindowServerMemory

Returns the amount of memory being used by a context.

```
NSInteger NSGetWindowServerMemory (
   NSInteger context,
   NSInteger *virtualMemory,
   NSInteger *windowBackingMemory,
   NSString **windowDumpString
);
```

Discussion
Calculates the amount of memory being used at the moment by the given *context*. If `NULL` is passed for *context*, the current context is used. The amount of virtual memory used by the current context is returned in the int pointed to by *virtualMemory*; the amount of window backing store used by windows owned by the current context is returned in the int pointed to by *windowBackingMemory*. The sum of these two numbers is the amount of the memory that this context is responsible for.

Calculating these numbers takes some time to execute; thus, calling this function in normal operation is not recommended.

If `nil` is not passed in for *windowDumpStream*, the information returned is echoed to the specified stream. This fact can be useful for finding out more about which windows are using up your storage.

Normally, `NSGetWindowServerMemory` returns 0. If `NULL` is passed for *context* and there's no current display context, this function returns –1.

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

Declared In
`NSGraphics.h`

NSHighlightRect

Highlights the specified rect by filling it with white.

```
void NSHighlightRect (
   NSRect aRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw.

Discussion
Highlights the rectangle referred to by *aRect*. Light gray becomes white, and white becomes light gray. This function must be called twice, once to highlight the rectangle and once to unhighlight it; the rectangle should not be left in its highlighted state. When not drawing on the screen, the compositing operation is replaced by one that fills the rectangle with light gray.

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

Declared In
`NSGraphics.h`

NSInterfaceStyleForKey

Returns an interface style value for the specified key and responder.

```
NSInterfaceStyle NSInterfaceStyleForKey (
   NSString *key,
   NSResponder *responder
);
```

Discussion
Used to determine an interface style based on a key and a responder, either of which may be `nil`. An `NSInterfaceStyle` value specifies the style in which an interface item, such as a button or a scroll bar, should be drawn. For example, a value of `NSMacintoshInterfaceStyle` indicates an item should be drawn in the Macintosh style. The values defined for `NSInterfaceStyle` are `NSNoInterfaceStyle`, `NSNextStepInterfaceStyle`, `NSWindows95InterfaceStyle`, and `NSMacintoshInterfaceStyle`. Note that this function never returns `NSNoInterfaceStyle`.

The interface style value returned by this function depends on several factors. If responder is not `nil` and if responder specifies an interface style other than `NSNoInterfaceStyle`, this function returns the responder's style, and *`key`* is ignored.

Otherwise, if *`key`* is not `nil` and there is an interface style for *`key`* specified by the defaults system, this function returns the interface style for *`key`* from the defaults system.

Finally, if *`key`* is `nil`, or if there is no interface style for *`key`* specified by the defaults system, this function returns the global interface style specified by the defaults system.

The defaults system allows an application to customize its behavior to match a user's preferences. You can read about the defaults system in the documentation for `NSUserDefaults`.

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSInterfaceStyle.h`

NSIsControllerMarker

Tests whether a given object is special marker object used for indicating the state of a selection in relation to a key.

```
BOOL NSIsControllerMarker (
   id object
);
```

Parameters

`Term`

Specify the object you want to check. This parameter can be `nil`.

Return Value
`YES` if the object is one of the designated controller markers or `NO` if it is not.

Discussion
This function helps you to create bindings between user interface elements and controller objects. The Application Kit predefines several special marker objects used as values for indicating selection state; currently these are `NSMultipleValuesMarker`, `NSNoSelectionMarker`, and `NSNotApplicableMarker`. These markers are typed as `id` and only exist for the purpose of indicating a state; they are never archived and cannot be used as object values in controls. You use this function to test whether a given object value is a marker, in which case it is not directly assignable to the object that is bound. This check is important, especially since additional markers may be added in the future.

See the `NSKeyValueBinding.h` header file for further details.

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

Related Sample Code
CoreRecipes

Declared In
`NSKeyValueBinding.h`

NSNumberOfColorComponents

Returns the number of color components in the specified color space.

```
NSInteger NSNumberOfColorComponents (
   NSString *colorSpaceName
);
```

Discussion
Returns the number of color components in the color space whose name is provided by `colorSpaceName`.

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

Declared In
`NSGraphics.h`

NSOpenGLGetOption

Returns global OpenGL options.

```
void NSOpenGLGetOption (
   NSOpenGLGlobalOption pname,
   GLint *param
);
```

Discussion
Returns in *`param`* the value of the global OpenGL parameter *`pname`*. The available options are enumerated by the `NSOpenGLGlobalOption` type.

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

Declared In
NSOpenGL.h

NSOpenGLGetVersion

Returns the NSOpenGL version numbers.

```
void NSOpenGLGetVersion (
   GLint *major,
   GLint *minor
);
```

Discussion
Returns by reference the major and minor version numbers of the NSOpenGL implementation. This function is not the same as the OpenGL version.

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

Declared In
NSOpenGL.h

NSOpenGLSetOption

Sets global OpenGL options.

```
void NSOpenGLSetOption (
   NSOpenGLGlobalOption pname,
   GLint param
);
```

Discussion
Sets the value of the global OpenGL parameter *pname* to *param*. The available options are enumerated by the `NSOpenGLGlobalOption` type.

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

Declared In
NSOpenGL.h

NSPerformService

Programmatically invokes a Services menu service.

```
BOOL NSPerformService (
   NSString *itemName,
   NSPasteboard *pboard
);
```

Parameters

itemName

Specifies a Services menu item, in any language. If the requested service is from a submenu of the Services menu, the value must contain a slash (for example, "Mail/Selection").

pboard

The pasteboard containing the data required by the service. This data must be present for the service to succeed. On output, this pasteboard contains the data returned by the service provider.

Return Value
`YES` if the service was successfully performed or `NO` if it was not.

Discussion
Use this function to programmatically invoke a service found in the application's Services menu.

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

Declared In
`NSApplication.h`

NSPlanarFromDepth

Returns whether the specified window depth is planar.

```
BOOL NSPlanarFromDepth (
   NSWindowDepth depth
);
```

Discussion
Returns `YES` if the specified window *depth* is planar and `NO` if it is not.

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

Declared In
`NSGraphics.h`

NSReadPixel

Reads the color of the pixel at the specified location.

```
NSColor * NSReadPixel (
   NSPoint passedPoint
);
```

Parameters

passedPoint

The pixel location to read, specified in the current coordinate system.

Return Value
The color of the pixel at the specified location.

Discussion
Because the *passedPoint* parameter is relative to the current coordinate system, if you wish to read a pixel from a specific view, you must convert points in the view's coordinate system to the current coordinate system before calling this function. Alternatively, you can lock focus on the view and then specify the pixel coordinate in the view's coordinate system.

When mapping the specified point to pixel boundaries, this method rounds to the nearest pixel. For more information on how coordinate points map to the underlying pixels, see Coordinate Systems and Transforms in *Cocoa Drawing Guide*.

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

Related Sample Code
Color Sampler
Monochrome Image

Declared In
NSGraphics.h

NSRectClip

Modifies the current clipping path by intersecting it with the passed rect.

```
void NSRectClip (
   NSRect aRect
);
```

Parameters

aRect
The rectangle to intersect with the current clipping rectangle.

Discussion
This function modifies the clipping path permanently. If you need to undo this modification later, you should save the current graphics state before calling this function and restore it once you are done.

A side effect of this function is that it clears the current Quartz 2D drawing path information. If you used Quartz 2D functions to create a drawing path in the current context, and you want to save that path information and use it later, you should transfer it to a `CGPathRef` opaque type before calling this function. If you are using only Cocoa to do your drawing, this behavior should not affect you.

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

Related Sample Code
Aperture Edit Plugin - Borders & Titles

Declared In
NSGraphics.h

NSRectClipList

Modifies the current clipping path by intersecting it with the passed rect.

```
void NSRectClipList (
   const NSRect *rects,
   NSInteger count
);
```

Parameters

rects

A pointer to an array of `NSRect` structures, which are combined and intersected with the current clipping path.

count

The number of rectangles in *rects*.

Discussion

This function modifies the clipping path permanently by generating a graphical union of the specified rectangles and then intersecting that union with the current clipping path. If you need to undo this modification later, you should save the current graphics state before calling this function and restore it once you are done.

A side effect of this function is that it clears the current Quartz 2D drawing path information. If you used Quartz 2D functions to create a drawing path in the current context, and you want to save that path information and use it later, you should transfer it to a `CGPathRef` opaque type before calling this function. If you are using only Cocoa to do your drawing, this behavior should not affect you.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSGraphics.h`

NSRectFill

Fills the passed rectangle with the current color.

```
void NSRectFill (
   NSRect aRect
);
```

Parameters

aRect

The bounding rectangle (in the current coordinate system) in which to draw.

Discussion

Fills *aRect* with the current color using the compositing mode `NSCompositeCopy`, which fills with the current color by copying the RGBA values. Use `NSRectFillUsingOperation` (page 52) to fill specifying a compositing mode.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
Dicey
FilterDemo
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSGraphics.h`

NSRectFillList

Fills the rectangles in the passed list with the current fill color.

```
void NSRectFillList (
   const NSRect *rects,
   NSInteger count
);
```

Parameters

rects
A pointer to an array of `NSRect` structures representing the rectangles to fill.

count
The number of rectangles in *rects*.

Discussion
Fills the specified rectangles with the current fill color using the compositing mode `NSCompositeCopy`.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

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

Declared In
`NSGraphics.h`

NSRectFillListUsingOperation

Fills the rectangles in a list using the current fill color and specified compositing operation.

```
void NSRectFillListUsingOperation (
   const NSRect *rects,
   NSInteger count,
   NSCompositingOperation op
);
```

Parameters

rects
A pointer to an array of `NSRect` structures representing the rectangles to fill.

count
The number of rectangles in the *rects* parameter.

op

The compositing operation to use when filling the rectangles.

Discussion
Fills a list of *count* rectangles with the current fill color, using the compositing operation *op*. For example, specifying `NSCompositeSourceOver` will blend with what's already been drawn.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

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

Declared In
`NSGraphics.h`

NSRectFillListWithColors

Fills the rectangles in the passed list with the passed list of colors.

```
void NSRectFillListWithColors (
   const NSRect *rects,
   NSColor **colors,
   NSInteger num
);
```

Parameters

rects

A pointer to an array of `NSRect` structures representing the rectangles to fill.

colors

A pointer to an array of `NSColor` objects. The number of color objects in this parameter must equal the number of rectangles in the *rects* parameter.

num

The number of rectangles in the *rects* parameter.

Discussion
Takes a list of *num* rectangles and a matching list of color objects. The first rectangle is filled with the first color, the second rectangle with the second color, and so on. There must be an equal number of rectangles and color values. The rectangles are composited using the `NSCompositeCopy` operator and the order in which the rectangles are filled cannot be guaranteed; therefore, overlapping rectangles may not draw as expected. This function alters the current color of the current graphics state, setting it unpredictably to one of the values passed in *colors*.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

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

Declared In
`NSGraphics.h`

NSRectFillListWithColorsUsingOperation

Fills the rectangles in a list using the specified colors and compositing operation.

```
void NSRectFillListWithColorsUsingOperation (
   const NSRect *rects,
   NSColor **colors,
   NSInteger num,
   NSCompositingOperation op
);
```

Parameters

rects

A pointer to an array of `NSRect` structures representing the rectangles to fill.

colors

A pointer to an array of `NSColor` objects. The number of color objects in this parameter must equal the number of rectangles in the *rects* parameter.

num

The number of rectangles in the *rects* parameter.

op

The compositing operation to use when filling the rectangles.

Discussion

Takes a list of *num* rectangles and a matching list of color values. The first rectangle is filled with the first color, the second rectangle with the second color, and so on. There must be an equal number of rectangles and color values. Each fill operation is performed using the compositing operation *op*. The rectangles should not overlap; the order in which they are filled cannot be guaranteed. This function alters the current color of the current graphics state, setting it unpredictably to one of the values passed in *colors*.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSGraphics.h`

NSRectFillListWithGrays

Fills the rectangles in the passed list with the passed list of grays.

```
void NSRectFillListWithGrays (
   const NSRect *rects,
   const CGFloat *grays,
   NSInteger num
);
```

Parameters

rects

A pointer to an array of `NSRect` structures representing the rectangles to fill.

grays

A pointer to an array of floating-point values in the range 0.0 to 1.0, where 0.0 represents absolute black and 1.0 represents absolute white and numbers in between are varying levels of gray. Values outside this range are clamped to 0.0 or 1.0.

num

The number of rectangles in the *rects* parameter.

Discussion
Takes a list of *num* rectangles and a matching list of gray values. The first rectangle is filled with the first gray, the second rectangle with the second gray, and so on. There must be an equal number of rectangles and gray values. The rectangles are composited using the `NSCompositeCopy` operator and the order in which the rectangles are filled cannot be guaranteed; therefore, overlapping rectangles may not draw as expected. This function alters the current color of the current graphics state, setting it unpredictably to one of the values passed in *grays*.

For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

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

Declared In
`NSGraphics.h`

NSRectFillUsingOperation

Fills a rectangle using the current fill color and the specified compositing operation.

```
void NSRectFillUsingOperation (
   NSRect aRect,
   NSCompositingOperation op
);
```

Parameters

aRect
The rectangle to fill with the current fill color.

op
The compositing operation to use when filling the rectangle.

Discussion
For a list of compositing operations and how you use them, see *Cocoa Drawing Guide*.

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

Related Sample Code
Cropped Image
GLChildWindowDemo
RGB Image
RGB ValueTransformers
Tinted Image

Declared In
`NSGraphics.h`

NSRegisterServicesProvider

Registers a service provider.

```
void NSRegisterServicesProvider (
   id provider,
   NSString *name
);
```

Parameters

provider

The object providing the service you want to register.

name

The unique name to associate with the service. This string is used to advertise the service to interested clients.

Discussion

Use this function to register custom services not directly related to your application.

You should not use this function to register the services provided by your application. For your application's services, you should use the `setServicesProvider:` method of `NSApplication`, passing a non-`nil` argument.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSApplication.h`

NSReleaseAlertPanel

Releases an alert panel.

```
void NSReleaseAlertPanel (
   id panel
);
```

Discussion

When you're finished with a panel created by a function such as `NSGetAlertPanel` (page 39), `NSGetCriticalAlertPanel` (page 40), or `NSGetInformationalAlertPanel` (page 41), you must free it by passing it to this function.

Note that the alert panel may not be deallocated immediately because it may have internal references that are released in a deferred way. You should not make the assumption that the alert panel is immediately removed from the application window list.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSPanel.h`

NSRunAlertPanel

Creates an alert panel.

```
NSInteger NSRunAlertPanel (
   NSString *title,
   NSString *msg,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   ...
);
```

Discussion
Creates and runs an alert panel (or dialog) with the title of *title*, the text of *msg*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*. See the description of `NSBeginAlertSheet` (page 18) for information on layout of buttons, default parameters, and possible return values. `NSRunAlertPanel` runs the panel in a modal event loop.

A Command-D key equivalent for the "Don't Save" button is provided, if one is found. The button titles are searched for the localized value for "Don't Save." If a match is found, that button is assigned a Command-D key equivalent, provided it is not the default button.

If you create a modal panel using `runModalForWindow:` or `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:`, you can assign the key equivalent yourself, using `setKeyEquivalent:` and `setKeyEquivalentModifierMask:`.

This function not only creates the panel; it also puts the panel onscreen and runs it using the `runModalForWindow:` method defined in the `NSApplication` class. This method sets up a modal event loop that causes the panel to remain onscreen until the user clicks one of its buttons. This function then removes the panel from the screen list and returns a value that indicates which of the three buttons the user clicked. For efficiency, this function creates the panel the first time it's called and reuses it on subsequent calls, reconfiguring it if necessary.

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

Related Sample Code
CocoaSpeechSynthesisExample
Quartz Composer WWDC 2005 TextEdit
SimpleCocoaApp
TextEditPlus
WhackedTV

Declared In
`NSPanel.h`

NSRunCriticalAlertPanel

Creates and runs a critical alert panel.

```
NSInteger NSRunCriticalAlertPanel (
   NSString *title,
   NSString *msg,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   ...
);
```

Discussion
Creates a critical alert panel that warns the user of some critical consequence of a requested action; the panel lets the user cancel the action and may allow the user to modify the action. It then runs the panel in a modal event loop.

The panel presented to the user is badged with a caution icon. Critical alerts should be used only as specified in the "Alerts" section of the Windows chapter of *Apple Human Interface Guidelines*.

The arguments for this function are the same as those for `NSRunAlertPanel` (page 53).

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

Related Sample Code
CIAnnotation
NewsReader
SimplePlayThru

Declared In
`NSPanel.h`

NSRunInformationalAlertPanel

Creates and runs an informational alert panel.

```
NSInteger NSRunInformationalAlertPanel (
   NSString *title,
   NSString *msg,
   NSString *defaultButton,
   NSString *alternateButton,
   NSString *otherButton,
   ...
);
```

Discussion
Creates an informational alert panel that provides information related to a requested action. It then runs the panel in a modal event loop.

The arguments for this function are the same as those for `NSRunAlertPanel` (page 53).

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

Declared In
`NSPanel.h`

NSSetFocusRingStyle

Specifies how a focus ring will be drawn.

```
void NSSetFocusRingStyle (
   NSFocusRingPlacement placement
);
```

Parameters

placement

Specifies how you want the focus ring to be drawn.

Discussion

Use `NSFocusRingAbove` to draw the focus ring over an image, use `NSFocusRingBelow` to draw the focus ring under text, and use `NSFocusRingOnly` if you don't have an image or text. For the `NSFocusRingOnly` case, fills a shape to add the focus ring around the shape.

Note that the focus ring may actually be drawn outside the view but will be clipped to any clipping superview or the window content view.

Availability

Available in Mac OS X v10.1 and later.

Related Sample Code

Clock Control

Dicey

TrackBall

Declared In

`NSGraphics.h`

NSSetShowsServicesMenuItem

Specifies whether an item should be included in Services menus.

```
NSInteger NSSetShowsServicesMenuItem (
   NSString *itemName,
   BOOL enabled
);
```

Discussion

Deprecated. This function simply returns 0.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSApplication.h`

NSShowAnimationEffect

Runs a system animation effect.

```
void NSShowAnimationEffect (
   NSAnimationEffect animationEffect,
   NSPoint centerLocation,
   NSSize size,
   id animationDelegate,
   SEL didEndSelector,
   void *contextInfo
);
```

Parameters

animationEffect

The type of animation you want to apply.

centerLocation

The location at which to show the animated image, specified in screen coordinates. The animation is centered on the point you specify.

size

The desired size of the animated image. Specify `NSZeroSize` to perform the animation at the default size.

animationDelegate

The object to notify when the animation completes. Specify `nil` if you do not need to be notified when the animation completes.

didEndSelector

The selector of *animationDelegate* to call when the animation completes. Specify `nil` if you do not need to be notified when the animation completes. If you specify a selector, the corresponding method should have the following signature:

```
- (void)animationEffectDidEnd:(void *)contextInfo;
```

contextInfo

A pointer to any optional information you want passed as a parameter to the selector in the *didEndSelector* parameter.

Discussion

This function runs one of the standard system animation effects, which includes display and sound. For example, you can use this function to display the puff of smoke effect. For a complete list of animation effects, see `NSAnimationEffect`.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`NSGraphics.h`

NSShowsServicesMenuItem

Specifies whether a Services menu item is currently enabled.

```
BOOL NSShowsServicesMenuItem (
   NSString *itemName
);
```

Discussion

Deprecated. This function simply returns `YES`.

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

Declared In
`NSApplication.h`

NSUnregisterServicesProvider

Unregisters a service provider.

```
void NSUnregisterServicesProvider(NSString *name);
```

Parameters

name
The name of the service you want to unregister.

Discussion
Use this function to unregister custom services not directly related to your application.

You should not use this function to unregister the services provided by your application. For your application's services, you should use the `setServicesProvider:` method of `NSApplication`, passing a `nil` argument.

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

Declared In
`NSApplication.h`

NSUpdateDynamicServices

Causes the services information for the system to be updated.

```
void NSUpdateDynamicServices (void);
```

Discussion
Used by a service-providing application to reregister the services it is willing to provide. To do this, you create a bundle with the extension "`.service`" and place it in the application's path or `~/Library/Services`. The content of the bundle is identical to a normal service bundle. You then call this function.

It is only necessary to call this function if your program adds dynamic services to the system.

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

Related Sample Code
SpotlightFortunes

Declared In
`NSApplication.h`

NSWindowList

Gets information about onscreen windows.

```
void NSWindowList (
   NSInteger size,
   NSInteger list[]
);
```

Discussion
Provides an ordered list of all onscreen windows. It fills *list* with up to *size* window numbers; the order of windows in the array is the same as their order in the window server's screen list (their front-to-back order on the screen). Use the count obtained by `NSCountWindows` (page 23) to specify the size of the array for this function.

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

Declared In
`NSGraphics.h`

NSWindowListForContext

Gets information about an application's onscreen windows.

```
void NSWindowListForContext (
   NSInteger context,
   NSInteger size,
   NSInteger list[]
);
```

Discussion
Provides an ordered list of onscreen windows for a particular application, identified by *context*, which is a window server connection ID. It fills *list* with up to *size* window numbers; the order of windows in the array is the same as their order in the window server's screen list (their front-to-back order on the screen). Use the count obtained by the `NSCountWindowsForContext` (page 24) function to specify the size of the array for this function.

Use of this function is discouraged as it may be deprecated in a future release.

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

Declared In
`NSGraphics.h`

Document Revision History

This table describes the changes to *Application Kit Functions Reference*.

Date	Notes
2008-11-19	Added descriptions for the NSDrawThreePartImage and NSDrawNinePartImage functions.
2008-10-15	Augmented description of NSApplicationMain.
2008-02-08	Updated thread safety information for the NSApplicationMain function.
2007-12-11	Made minor enhancements throughout. Added description of NSAccessibility function NSAccessibilityUnignoredAncestor.
2007-07-16	Removed descriptions of deprecated functions no longer appearing in public header files.
2006-05-23	First publication of this content as a separate document.

Document Revision History

Index

N